INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Models and complexity results for performance and energy optimization of concurrent streaming applications

Anne Benoit — Paul Renaud-Goud — Yves Robert

N° 7589

April 2011

Distributed and High Performance Computing

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--7589--FR+ENG

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche GRENOBLE - RHÔNE-ALPES

Models and complexity results for performance and energy optimization of concurrent streaming applications

Anne Benoit , Paul Renaud-Goud , Yves Robert

Theme : Distributed and High Performance Computing
Équipe-Projet GRAAL

Rapport de recherche n° 7589 — April 2011 — 35 pages

Abstract: In this report, we study the problem of finding optimal mappings for several independent but concurrent workflow applications, in order to optimize performance-related criteria together with energy consumption. Each application consists in a linear chain graph with several stages, and processes successive data sets in pipeline mode, from the first to the last stage. The problem is to decide which processors to enroll, at which speed (or mode) to use them, and which stages they should execute. There is a clear trade-off to reach, since running faster and/or more processors leads to better performance, but energy consumption is then very high. Energy savings can be achieved at the price of a lower performance, by reducing processor speeds or enrolling fewer resources. We study the problem complexity on different target execution platforms, ranking from fully homogeneous platforms to fully heterogeneous ones. We consider three mapping strategies: (i) one-to-one mappings, where a processor is assigned a single stage; (ii) interval mappings, where a processor may process an interval of consecutive stages of the same application; and (iii) general mappings, which are fully arbitrary, i.e., a processor may process stages of several distinct applications. Finally, we compare two different models for the computation of the latency, which is the time elapsed between the beginning and the end of the execution of a given data set: with the PATH model, it is computed as the length of the path taken by this data set, while with the WAVEFRONT model, each data set progresses concurrently within a period. For all platform types, all mapping strategies and both latency models, we establish the complexity of several multi-criteria optimization problems, whose objective functions combine period, latency and energy criteria. In particular, we exhibit instances where the problem is NP-hard with concurrent applications, while it can be solved in polynomial time for a single application, and instances whose problem complexity depends upon the latency model.

Key-words: mapping, concurrent streaming applications, heterogeneous platforms, complexity results, resource sharing, energy, latency, period.

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Modèles et résultats de complexité pour l'optimisation de l'énergie et des performances d'applications concurrentes pipelinées

Résumé : Dans ce rapport, nous étudions le problème de trouver des placements pour plusieurs applications pipelinées concurrentes, dans le but de minimiser à la fois la performance des applications et la consommation d'énergie. Nous considérons plusieurs plates-formes d'exécution dont le degré d'hétérogénéité varie, ainsi que plusieurs stratégies de placement. Finalement, nous introduisons deux modèles d'exécution pour calculer la latence. Pour chacune des variantes du problème, nous établissons la complexité de plusieurs problèmes d'optimisation multi-critères.

Mots-clés : applications pipelinées, plates-formes hétérogènes, résultats de complexité, partage de ressources, énergie, latence, période.

Contents

1 Introduction

In this report, we aim at optimizing the parallel execution of several pipelined applications that execute concurrently on a given platform. We focus in this work on pipelined applications whose structure is a linear chain of tasks. Such applications are ubiquitous in streaming environments, as for instance video and audio encoding and decoding, DSP applications, image processing, and so on [13, 16, 26, 30, 31]. Furthermore, the regularity of these applications render them amenable to a high-level parallel programming approach based on algorithmic skeletons [12, 23]. Skeletons ease the task of the application developer and make it easy to tailor his/her specific problem to a target platform.

In a linear pipelined application, a series of data sets enters the input stage and progresses from stage to stage until the final result is computed. Each stage corresponds to a distinct task and has its own communication and computation requirements: it reads an input from the previous stage, processes the data and outputs a result to the next stage. Each data set is input to the first stage, and final results are output from the last stage. The pipeline operates in synchronous mode: after a transient behavior due to the initialization delay, a new data set is completed every period. Mapping such applications onto parallel platforms is a challenging problem, that becomes even more difficult when platforms are heterogeneous (nowadays a standard assumption). Another level of difficulty is added when considering several independent applications which are executed concurrently on the platform and that compete for available resources.

The objective is to minimize the energy consumption of the whole platform, while satisfying given performance-related bounds on the period and latency of each application. This problem has recently become a critical problem, both for economic and environmental reasons [22]. The Green500 list (`www.green500.org`) provides rankings of the most energy-efficient supercomputers in the world, therefore raising awareness about power consumption. The multi-criteria approach targets a trade-off between the users and the platform manager. The formers have specific requirements for their applications, while the latter has crucial economical and environmental constraints. Indeed, the energy saving problem is becoming increasingly important, not only because of the sole cost of energy, but also because of the cost of cooling systems and related infrastructures. To help reduce energy costs, modern computing centers provide multi-modal processors: each processor has a discrete number of predefined speeds (or modes), which correspond to different voltages that the processor can be subjected to. This approach is the Dynamic Voltage Scaling (DVS) technique; the slowest modes correspond to a lower voltage, and hence a slower execution speed. The power consumption is the sum of a static part (the cost for a processor to be turned on: power can be saved by shutting processors down – ON/OFF technique) and a dynamic part. This dynamic part is a strictly convex function of the processor speed, so that the execution of a given amount of work costs more energy if a processor runs in a higher mode [17, 19]. On the one side, faster modes (i.e., higher voltages) allow for fulfilling the performance criteria, and on the other side, they lead to a higher energy consumption.

The main performance-oriented criteria for pipelined applications are period and latency [6, 7, 24, 25, 28–30]. The period of an application is the inverse of the throughput, i.e., it corresponds to the time interval between the arrival of two consecutive data sets. The period is dictated by the critical resource: it is equal

to the longest cycle time of a processor. For instance under a strict one-port communication model with no overlap of communications and computations, it is the sum of the time to perform all incoming communications, the time to perform all outgoing communications, and the total computation time. With overlap, we simply replace the sum of these three terms by their maximum. In some cases, the period is fixed by the applicative setting, and we must ensure that data sets are processed fast enough so that there is no accumulation of data sets in the pipeline. The latency of an application 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 to process the data set entirely. For streaming applications, there are several approaches to compute the latency. The most accurate model is the PATH model, in which the latency is computed as the length of the path taken by any data set. With the WAVEFRONT model, we rather consider that each data set progresses concurrently within a period, and the latency is then a multiple of the period, as suggested by Hary and Özgüner in [16].

The two performance criteria alone already are antagonistic. The smallest latency is obtained when no communication occurs, i.e., when the same (fastest) processor executes all the stages of an application. However, such a mapping may well exceed the bound on the period, since the same processor must process an entire application. Moreover, when several applications run concurrently, the scheduler must decide which resources to select and assign to each application, so that all users receive a fair share of the platform.

Adding energy consumption as a third criterion renders everything even more complex. Obviously, energy is minimized by enrolling a single processor for all applications, namely the one with the smallest speed available; but such a mapping would most certainly exceed period and latency bounds.

Our goal is to execute all applications efficiently while minimizing the energy consumed. Unfortunately, the goals of low power consumption and efficient scheduling are contradictory. Indeed, period and/or latency can be minimized by using more energy to speed up processors, while energy can be minimized by reducing processor speeds, hence performance-related objectives. How to deal with these contradictory objective functions? In traditional approaches, one would form a linear combination of the different objectives and treat the result as the new objective to be optimized. But is it natural for the user to maximize the quantity 0.7P + 0.3E, where P is the period and E the energy? Since criteria are very different in nature, it does not make much sense for a user to make a linear combination of them. Thus we advocate the use of multi-criteria mappings with thresholds. Now, each criteria combination can be handled in a natural and meaningful way: one single criterion is optimized, under the condition that a threshold is enforced for all other criteria. This leads to two interesting questions. If we fix energy, we get the *laptop problem*, which asks "What is the best schedule achievable using a particular energy budget, before battery becomes critically low?" Fixing schedule quality gives the *server problem*, which asks "What is the least energy required to achieve a desired level of performance?"

The optimization problem can then be stated as follows: given a set of applications and a computational platform, which stage to assign to which processor? We consider three different mapping strategies: *one-to-one* mappings, for which each application stage is allocated to a distinct processor; *interval* mappings,

where each participating processor is assigned an interval of consecutive stages of the same application; and *general* mappings which are fully arbitrary. These mapping strategies have been widely used in the literature when mapping one single application (see [6,24,25]), and we extend them naturally to map of several concurrent applications.

We target three different platform types: *fully homogeneous* platforms have identical processors and interconnection links; *communication homogeneous* platforms have identical links but different-speed processors, thus introducing a first degree of heterogeneity; and finally, *fully heterogeneous* platforms, with different-speed processors and different capacity links, constitute the most difficult problem instance.

The report is organized as follows. We first review related work in Section 2. Then, we illustrate and motivate the problem with a simple example in Section 3. The framework is described in Section 4. The next two sections constitute the heart of the report: we assess the complexity of all problem instances. In Section 5, we establish the complexity of mapping problems with the PATH latency model, while we investigate the complexity with the WAVEFRONT latency model in Section 6. We conclude in Section 7.

2 Related work

The problem of mapping a single linear chain application onto parallel platforms in order to minimize latency and/or period has already been widely studied, in particular on homogeneous platforms (see the pioneering papers [24] and [25]) and later for heterogeneous platforms (see [6, 7]), considering the PATH latency model. These results focus on the mapping of one single application, while we add the complexity of satisfying several users who each have different requirements for their applications. We were able to extend polynomial time algorithms to this multi-application setting, and to exhibit cases in which the problem becomes NP-hard because of this additional difficulty. Of course, problem instances which were already NP-hard with a single application remain difficult with several concurrent applications.

Moreover, we consider a new and important objective function, namely energy minimization, and this is the first study (to the best of our knowledge) which combines performance-related objectives with energy in the context of pipelined applications. As expected, combining all three criteria (period, latency and energy) leads to even more difficult optimization problems: the problem is NP-hard even with a single application on a fully homogeneous platform (for interval mappings with the PATH latency model).

In order to adjust energy consumption, we use the Dynamic Voltage Scaling (DVS) technique. DVS has been extensively studied in several papers, for mapping onto a single-core processor, a multi-core processor, or a set of processors.

Slack reclamation techniques are used for frame-based hard real-time embedded system in [32]: a set of independent tasks, provided with their WCEC (Worst Case Execution Cycle) and sharing a common deadline, has to be mapped onto a processor. If a task needs less cycles than its WCEC, the dynamically obtained slack allows the processor to run at a lower frequency and therefore to spare energy. This work is extended in [33], where the energy model includes time and energy penalties when the processor frequency is changing. Those

transition overheads are also taken into account in [2], but tasks are interdependent.

Then [11] maps applications which consists of a program modeled with a sequential part and another part which can be parallel, onto a multi-core processor. Bunde [9] focuses on the problem of offline scheduling unit time tasks with release dates, while minimizing the makespan or the total flow time on one processor. He extends this work from one processor to multi-processors.

Authors in [10] study the problem of scheduling real-time tasks on two heterogeneous processors. They provide a FPTAS to derive a solution very close to the optimal energy consumption with a reasonable complexity, while in [15], the authors design heuristics to map a set of dependent tasks with deadlines onto a set of homogeneous processors, with the possibility of changing a processor speed during the execution of a task. [18] proposes a greedy algorithm based on affinity to assign frame-based real-time tasks, and then they re-assign them in pseudo-polynomial time when any processing speed can be assigned for a processor. In [21], leakage energy is the focus for mapping applications represented as DAGs. In [27], the authors are interested about scheduling task graphs with data dependencies while minimizing the energy consumption of both the processors and the inter-processor communication devices, while assuming the communication times are negligible compared to the computation times.

All these problems are quite different from ours, since we focus on pipelined applications of infinite duration, thus considering power instead of total energy consumption. Due to the streaming nature of the applications, we do not allow for changing the processor speed during execution.

3 Motivating example

In this example, we have two applications and three processors, as shown on Figure 1. The first stage of App_1 computes 3 operations, and then sends a data of size 3 to the second stage; the second stage first receives a data of size 3, then computes 2 operations, and finally sends a data of size 1, and so on. If both stages are assigned to the same processor, there is no communication cost to pay; otherwise this cost depends on the communication volume (3 between the first and the second stage in this case), and on the link bandwidth between the corresponding processor pair. All communication link bandwidths are set to 1. For the computational platform, each processor has two execution modes. For instance, P_1 can process 3 operations per time unit in its first mode, and 6 in its second one, against 6 or 8 for P_2, and 1 or 6 for P_3.

We compute the global period as follows: $T = \max(T_1, T_2)$, where T_i is the period of the i^{th} application ($i = 1, 2$). The global latency is defined in a similar way, as the maximum of the latency achieved by all applications. The energy

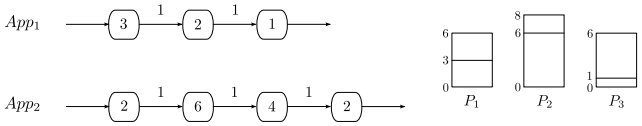

Figure 1: Example with two applications and three multi-modal processors.

consumption of a processor is equal to the square of its speed, which is quite a realistic assumption (see Section 4.5 for more details on the model for energy consumption). Note that when the energy is not a criterion to minimize, all processors can run in their higher modes (as fast as possible), because this can only improve the performance-related criteria (period and latency). In this case, either a processor is used at its fastest speed, or it is turned off.

3.1 Interval mappings

First we restrict to interval mappings, where a processor can be assigned only a set of consecutive stages of a single application.

In order to minimize the period without energy constraints, we map the whole first application onto processor P_3, the first half of the second application onto processor P_2, and the rest onto processor P_1. The period is then:

$$\max\left(\tfrac{3+2+1}{6}, \max\left(\tfrac{2+6}{8}, \tfrac{1}{1}, \tfrac{4+2}{6}\right)\right) = 1\ . \tag{1}$$

Equation (1) reads as follows: we compute the cycle-time of each processor as the maximum time spent for incoming communications, computations, and outgoing communications, thus considering a model in which communications and computations are overlapped. We then take the maximum of these quantities to derive the period. There is only one communication to pay in the second application since it is split between two processors. Note that the cycle-time of each processor is exactly 1 and there is no idle time on computation, thus it is not possible to achieve a better period: this mapping is optimal for the period minimization problem.

The minimum latency is obtained by removing all communications and using the fastest processors. A mapping that returns the optimal latency (in the absence of other criteria) is for instance the one which maps the first application on P_1 and the second application on P_2, thus achieving a global latency of:

$$\max\left(\tfrac{3+2+1}{6}, \tfrac{2+6+4+2}{8}\right) = 1.75\ . \tag{2}$$

In Equation (2), we simply compute the longest execution path for each application following the Path latency model. The period of each application is, in this case, equal to its latency, and the Wavefront model returns the same latency (one single period for the execution of a data set). The bottleneck is the second application, and we cannot achieve a better latency since we pay no communication and use the fastest processor for this application. This latency is thus optimal.

The minimum energy is obtained when we use fewer processors, each running in its slowest mode. Since we assume that a processor cannot be assigned stages of two different applications, two processors are required in the example. For instance, we can map the first application on P_1 running in its lowest mode and the second application on P_3 running in its lowest mode too, thus achieving an energy of $3^2 + 1^2 = 10$. This is the minimum energy consumption required to run both applications. We observe that the period is then:

$$\max\left(\tfrac{3+2+1}{3}, \tfrac{2+6+4+2}{1}\right) = 14\ .$$

As expected, running at a slower mode to save energy leads to poorer performances. Trade-offs must be found when considering several antagonistic optimization criteria.

For instance, if we try to minimize the energy consumption under the constraint that the period is not greater than 2, we can use the first mode of each processor. Then the first application is mapped onto P_1, the first three stages of the second application are mapped onto P_2 and its last stage is mapped onto P_3. The global period is 2, and the consumed energy is $3^2 + 6^2 + 1^2 = 46$. This may be quite a reasonable compromise between energy and period: indeed, with the mapping minimizing the period (period of 1), the energy consumption was $6^2 + 8^2 + 6^2 = 136$. With this mapping, the latency model impacts the result. With the PATH model, we compute the longest path followed by a data set, it is $\max\left(\frac{3+2+1}{3}, \frac{2+6+4}{6} + \frac{1}{1} + \frac{2}{1}\right) = 5$, while with the WAVEFRONT model, it takes three periods for a data set to be computed by the second application, leading to a latency of $3 \times 2 = 6$.

3.2 General mappings

With general mappings, it is possible to assign any subset of stages to the processors. For instance, we consider the mapping in which the first stage of application one, and the second and third stages of application two, are all mapped onto the second processor, running at speed 6. The other stages are mapped onto the first processor, running at speed 3.

The energy consumption is then $3^2 + 6^2 = 45$. For the period, we take the maximum between the periods of both processors, accounting both for computation and communication costs:

$$\max\left(\tfrac{1+1}{1}, \tfrac{2+1+2+2}{3}, \tfrac{1}{1}, \tfrac{3+6+4}{6}, \tfrac{1+1}{1}\right) = \tfrac{7}{3} .$$

Note that there are two communications from P_2 to P_1: one which corresponds to the communication in the first application between the first and the second stage, and one in the second application between the third and the fourth stage. For the computation of the latency with the PATH model, it is necessary to decide in which order these communications occur. If we start with the communication in the first application, the latency is computed as follows:

$$\max\left(\tfrac{3}{6} + \tfrac{1}{1} + \tfrac{2+1}{3}, \tfrac{2}{3} + \tfrac{1}{1} + \tfrac{6+4}{6} + \tfrac{1}{1} + \tfrac{1}{1} + \tfrac{2}{3}\right) = 6 .$$

There is one time unit of idle time in the computation of the latency of the second application, which corresponds to the communication from P_2 to P_1 in the first application. The latency can be reduced to 5 if we change the order of communications. Actually, for general mappings, even if the mapping is fixed, it is NP-hard to decide in which order communications should be executed in order to minimize the latency with the PATH model [1].

Because of this observation, we consider the WAVEFRONT model when dealing with general mappings. This model was introduced by Hary and Özgüner in [16], and it is widely used in real-time systems. Note that the WAVEFRONT model assumes a full overlap of communications and computations. In the example, the latency is still dictated by the second application: this application needs 5 periods to execute a whole data set. The WAVEFRONT latency is therefore $5 \times \frac{7}{3} \approx 11.66$.

4 Framework

We start with a formal description of the applicative framework (Section 4.1) and the target execution platform (Section 4.2). Next in Section 4.3, we introduce and motivate the mapping strategies. We are then ready to formally describe the performance objective criteria (period and latency) in Section 4.4, and then to finally discuss the energy model in Section 4.5.

4.1 Applicative framework

We consider A independent application workflows ($A \geq 1$) to be executed concurrently; each application operates on a collection of data sets that are executed in a pipelined fashion. For $1 \leq a \leq A$, let n_a be the number of stages of application a, and $N = \sum_{a=1}^{A} n_a$ be the total number of stages. For $1 \leq k \leq n_a$, w_a^k is the computation requirement of $\mathcal{S}_a^k$, the k^{th} stage of application a. For $1 \leq k < n_a$, δ_a^k is the size of the output data of $\mathcal{S}_a^k$.

4.2 Target platform

The target platform is composed of p processors, which are fully interconnected; there is a bidirectional link $P_u \leftrightarrow P_v$ between any processor pair P_u and P_v, of bandwidth $b_{u,v}$.

We use a linear cost model for communications; it takes $X/b_{u,v}$ time units to send (resp. receive) a message of size X to (resp. from) P_v. With the mapping rules that we enforce (see Section 4.3 below), it turns out that a processor never has to perform two concurrent ingoing nor outgoing communications: at any time-step, a processor is involved in at most one send, one computation and one receive. However, these three operations can either be parallel (as in the example of Section 3) or serialized. With parallel operations, we have the *overlap* model that corresponds to multi-threaded communication libraries such as MPICH2 [20]. With sequential operations, we have the *no-overlap* model that is well-suited to single-threaded programs.

Processors are multi-modal: every processor P_u is associated with a set of speeds $\mathsf{S_u} = \{s_{u,1}, \ldots, s_{u,m_u}\}$. During the mapping process, we need to choose one speed in $\mathsf{S_u}$ for each processor P_u that is enrolled, and this speed is fixed during the whole execution.

Then we classify particular cases which are important, both from a theoretical and practical perspective. *Fully homogeneous* platforms, also called *speed homogeneous*, have identical processors (all processors have a common speed set: $\mathsf{S_u} = \mathsf{S}$) and homogeneous communication devices ($b_{u,v} = b$ for all link bandwidths). They represent typical parallel machines. *Communication homogeneous* platforms, also called *speed heterogeneous*, are still interconnected with homogeneous communication devices, but they may have processors with different speed sets ($\mathsf{S_u} \neq \mathsf{S_v}$). They correspond to networks of workstations with plain TCP/IP interconnects or other LANs. *Fully heterogeneous platforms* are the most general, fully heterogeneous architectures. Hierarchical platforms made up with several clusters interconnected by slower backbone links can be modeled this way.

4.3 Mapping strategies and scheduling

We consider three mapping strategies. *One-to-one* mappings obey the simplest rule: each application stage is allocated to a distinct processor. While easier to optimize and implement, this rule may be unduly restrictive, and is likely to pay high communication costs. Obviously, it also requires that $p \geq N$, thereby limiting its applicability to larger platforms (or fewer and smaller applications). A natural extension is to search for *interval* mappings, where each participating processor is assigned an interval of consecutive stages. Intuitively, assigning several consecutive stages to the same processors will increase their computational load, but may well dramatically decrease communication requirements. Interval mappings have been widely used in the literature, see [6, 24, 25, 30, 31] among others. We point out that both one-to-one and interval mappings forbid any processor sharing, or re-use, across applications. These mappings are relevant in practice, for instance if we envision a computer center where applications, or jobs, cannot share resources because of security rules or of batch-assignment procedures. The goal of the platform manager is to secure an efficient (albeit concurrent) execution for each application (performance-related criteria) while minimizing the energy consumption of the whole platform.

We also introduce *general* mappings that allow any processor to execute any number of stages, consecutive or not, taken from one or several applications. Such mappings are likely to lead to a better resource utilization throughout the platform.

4.4 Performance optimization criteria

We are now ready to formally define the *period* and the *latency* of the applications. We start with one-to-one and interval mappings with no processor sharing, and then we discuss the impact of processor sharing on the metrics.

4.4.1 Without processor sharing

For one-to-one and interval mappings, since there is no processor sharing, we can focus on a single application.

Formally, an interval mapping is a partition of the set of stages $\mathcal{S}^1$ to $\mathcal{S}^n$ into m intervals $I_j = [d_j, e_j]$ such that $d_j \leq e_j$ for $1 \leq j \leq m$, $d_1 = 1$, $d_{j+1} = e_j + 1$ for $1 \leq j \leq m-1$ and $e_m = n$. Then, the function $\mathsf{al} : [1, n] \mapsto [1, p]$ associates a processor number to each stage number. In a one-to-one mapping, this function is a one-to-one assignment. In an interval mapping, for $1 \leq j \leq m$, the whole interval I_j is mapped onto the same processor $P_{\mathsf{al}(d_j)}$, i.e., for $d_j \leq i \leq e_j$, $\mathsf{al}(i) = \mathsf{al}(d_j)$. Also, two intervals (from the same application or from two different applications) cannot be mapped onto the same processor, i.e., for $1 \leq j, j' \leq m$, $j \neq j'$, $\mathsf{al}(d_j) \neq \mathsf{al}(d_{j'})$.

The period of this single application is expressed in the overlap model as:

$$T^{(overlap)} = \max_{j \in \{1,\ldots,m\}} \left(\max \left(\frac{\delta^{d_j - 1}}{b_{\mathsf{al}(d_j-1),\mathsf{al}(d_j)}}, \frac{\sum_{i=d_j}^{e_j} w^i}{s_{\mathsf{al}(d_j)}}, \frac{\delta^{e_j}}{b_{\mathsf{al}(d_j),\mathsf{al}(e_j+1)}} \right) \right), \quad (3)$$

with $\delta^{d_1 - 1} = \delta^{e_m} = 0$ for the boundaries.

The maximum in the previous expression is replaced by a sum when considering the no-overlap model, since all operations are serialized. The period is then:

$$T^{(no-overlap)} = \max_{j\in\{1,\ldots,m\}} \left(\frac{\delta^{d_j-1}}{b_{\mathsf{al}(d_j-1),\mathsf{al}(d_j)}} + \frac{\sum_{i=d_j}^{e_j} w^i}{s_{\mathsf{al}(d_j)}} + \frac{\delta^{e_j}}{b_{\mathsf{al}(d_j),\mathsf{al}(e_j+1)}} \right). \quad (4)$$

The latency is the time to process a single data entirely, so it is identical in both communication models, and computed with the PATH model:

$$L = \sum_{j=1}^{m} \left(\sum_{i=d_j}^{e_j} \frac{w^i}{s_{\mathsf{al}(d_j)}} + \frac{\delta^{e_j}}{b_{\mathsf{al}(d_j),\mathsf{al}(e_j+1)}} \right), \quad (5)$$

with $\delta^{e_m} = 0$ for the boundary.

Again, the simplicity of Equations (3), (4) and (5) is a very useful property of interval mappings, and greatly simplifies the solution of multi-criteria problems.

These are the period and latency of one single application, and we need to define a global period and latency function to be optimized. The simplest approach is to minimize $X = \max_{a\in\{1,\ldots,A\}}(X_a)$, where X_a is the period or latency of application a, for $a \in \{1,\ldots,A\}$, as in the example of Section 3. However, the concurrent applications can be of completely different nature and/or economic value, so that their periods or latencies are not always comparable. Therefore we aim at minimizing

$$X = \max_{a\in\{1,\ldots,A\}} W_a \times X_a, \quad (6)$$

where $W_a > 0$ is a weight associated to each application and X_a is the period or latency of application a, for $a \in \{1,\ldots,A\}$. W_a can be 1 (we retrieve a simple maximum) or a priority ratio (fixed by the platform manager and/or paid by the user). We can also let $W_a = 1/X_a^*$, where X_a^* is the objective function computed when the application is executed alone on the platform; in this case $W_a \times X_a$ represents the slowdown factor of application a, and X corresponds to the maximum stretch [3].

4.4.2 With resource sharing

If we keep the classical latency definition (PATH model) and consider general mappings, it leads to intricate scheduling problems for period/latency bi-criteria problems. Basically, even when the mapping is given, scheduling the execution is a problem of combinatorial nature (it is NP-complete, see [1]). With general mappings, a processor typically has several incoming and/or outgoing communications, and it is difficult to orchestrate these operations so as to minimize conflicting objectives such as period and latency. This holds true both for the overlap and no-overlap models.

Therefore, when considering resource sharing, we focus in this report on the problem in which bounds on period and latency are fixed by the application designer, and we relax the definition of the latency using the approach of Hary and Özgüner [16], that we call the WAVEFRONT model. Instead of computing the longest path, we approximate the latency L as $L = (2m-1)P$, where P is the period, i.e., the rate at which data sets enter the system, and m is

the number of intervals of consecutive stages mapped onto a same processor in the mapping. A processor change occurs each time when a stage and its successor are not mapped onto the same processor, i.e., $m-1$ times. The intuition is that the whole application is executed synchronously, and each data set progresses concurrently within a period. With m successive computations and $m-1$ processor changes (i.e., communications), each data set traverses the platform within $2m-1$ periods.

The mapping is an allocation function, which associates a processor number to each stage number, as well as a speed at which each processor is running. For general mappings with processor reuse, we must carefully decide how the speed of each processor is shared among all stages it is assigned to. Similarly, a communication link or processor network card may be involved in several communications, which implies to sharing bandwidths and card capacities, too. Hence the question is the following: given the mapping, and a threshold period P_a and latency L_a for each application $a \in \{1, \ldots, A\}$, is it possible to determine which fraction of computing and communicating resources to assign to each operation so that all period and latency thresholds are met?

Since we consider the WAVEFRONT latency model, one period is accounted for each computation of an interval of stages and for each inter-processor communication. We observe that given the mapping, we know m_a, the number of intervals (m_a-1 processor changes), for each application a. We can thus check immediately whether the bounds on the latency are respected, i.e., $(2m_a-1)P_a \leq L_a$ for $a \in \{1, \ldots, A\}$.

Now for the periods, the key idea is to distribute platform resources parsimoniously, and to allocate only the needed CPU fraction to each computation, and the needed bandwidth fraction to each communication, so that the period constraint is fulfilled. The mapping is valid if neither processor speeds, nor link bandwidths, nor network card capacities are exceeded. First, we merge consecutive stages $[\mathcal{S}_a^i, \ldots, \mathcal{S}_a^j]$ of application a mapped onto a same processor as one single coalesced stage $\hat{\mathcal{S}}_a^k$, with computing cost $\hat{w}_a^k = \sum_{k'=i}^{j} w_a^{k'}$, and output communication cost $\hat{\delta}_a^k = \delta_a^j$. The transformed application now has exactly m_a stages. In the following, stage $\hat{\mathcal{S}}_a^k$ corresponds to the k-th stage of the transformed application a, for $1 \leq k \leq m_a$.

As for computations, consider a processor $\mathcal{P}_u$ and an application a. We define $\mathcal{K}_a^u$ such that $k \in \mathcal{K}_a^u$ if and only if $\hat{\mathcal{S}}_a^k$ is processed by processor $\mathcal{P}_u$; $\mathcal{K}_a^u$ is the set of stages of (transformed) application a processed by $\mathcal{P}_u$. Then, for all a and u, and for each $k \in \mathcal{K}_a^u$, we allocate the speed fraction $s_{a,u}^k = \hat{w}_a^k / P_a$ for $\mathcal{P}_u$ to execute $\hat{\mathcal{S}}_a^k$.

Similarly for communications, we define $\mathcal{K}_a^{u,v}$ such that $k \in \mathcal{K}_a^{u,v}$ if and only if $\hat{\mathcal{S}}_a^k$ is processed by $\mathcal{P}_u$ and $\hat{\mathcal{S}}_a^{k+1}$ is processed by $\mathcal{P}_v$, i.e., there is a communication to pay between $\mathcal{P}_u$ and $\mathcal{P}_v$. Note that $u \neq v$, otherwise stages $\hat{\mathcal{S}}_a^k$ and $\hat{\mathcal{S}}_a^{k+1}$ would have been merged as a single stage. Formally, $k \in \mathcal{K}_a^{u,v} \Leftrightarrow k \in \mathcal{K}_a^u$ and $k+1 \in \mathcal{K}_a^v$. Then we allocate the bandwidth fraction $b_{a,u,v}^k = \hat{\delta}_a^k / P_a$ to the communication.

The period of each application can be respected if and only if all the following inequalities are satisfied. There might be some spare speed and bandwidth if

these are strict inequalities, and resources are fully utilized in case of equalities:

- $\forall 1 \leq u \leq p$, $\sum_{a=1}^{A} \sum_{k \in \mathcal{K}_a^u} s_{a,u}^k \leq s_u$, $\sum_{v=1}^{p} \sum_{a=1}^{A} \sum_{k \in \mathcal{K}_a^{u,v}} b_{a,u,v}^k \leq B_u^{out}$, $\sum_{v=1}^{p} \sum_{a=1}^{A} \sum_{k \in \mathcal{K}_a^{v,u}} b_{a,v,u}^k \leq B_u^{in}$;
- $\forall 1 \leq u, v \leq p,\ u \neq v,\ \sum_{a=1}^{A} \left(\sum_{k \in \mathcal{K}_a^{u,v}} b_{a,u,v}^k + \sum_{k \in \mathcal{K}_a^{v,u}} b_{a,v,u}^k \right) \leq b_{u,v}$.

Note that we can consider mappings without reuse with this latency model. In this case, if we transform each application a as explained above, the allocation function of stages $\hat{\mathcal{S}}_a^k$ (for $1 \leq a \leq A$ and $1 \leq k \leq m_a$) is a one-to-one function: each coalesced stage is allocated onto a distinct processor. It becomes then much easier to check the validity of the mapping, since each processor is only handling one single stage, receiving input data from one single other processor, and sending output data to one single other processor.

4.5 Energy model

The energy consumption of the platform is defined as the sum of the energy $E(u, \ell)$ consumed by each processor $\mathcal{P}_u$ enrolled in the mapping in mode ℓ. We assume that $E(u, \ell)$ consists of a static part and of a dynamic part. The static part $E_{stat}(u)$ is the static cost for a processor to be in service, and does not depend on the speed $s_{u,\ell}$ at which the processor is running. However, the static energy is consumed only in mode $\ell \neq 0$ (otherwise, the processor is inactive, and not enrolled in the mapping). On the contrary, the dynamic part $E_{dyn}(u, \ell)$ is of the form $E_{dyn}(u, \ell) = s_{u,\ell}^{\alpha}$, where $\alpha > 1$ is an arbitrary rational number. It is sometimes assumed that $\alpha = 2$ [19], as we did in the example of Section 3, but all our results hold for any value of α. Finally, for $\ell \neq 0$, we have $E(u, \ell) = E_{stat}(u) + E_{dyn}(u, \ell)$, while $E(u, 0) = 0$.

The energy $E(u, \ell)$ is an energy consumed per time unit, so we could also speak of dissipated power. Note that it is mandatory to minimize energy consumption per time unit, because the execution of streaming applications with arbitrarily many data sets may last for an unbounded amount of time. Hence we always consider a combination of energy and period objective criteria, because the latency by its own takes only one single data set into account, and does not reflect a pipelined execution.

5 Complexity results with the Path model

In this section, we consider the Path model for the computation of the latency, and therefore we restrict the study to one-to-one and interval mappings with no resource sharing. General mappings with resource sharing are investigated in Section 6.

In the following, **proc-hom** denotes identical speed processors while **proc-het** represents heterogeneous processors; **com-hom** means identical communication links, while they differ for **com-het**. We also report results for the case **special-app**, which corresponds to applications whose stages are all identical (all w_a^k are equal), and no communication cost is paid (all δ_a^k are equal to 0).

We start with the mono-criterion problems of period or latency minimization in Sections 5.1 and 5.2. In these cases, we do not consider energy minimization issues, and therefore we can systematically run processors at their highest speed,

and thus use classical results established in a context with no energy. Then we investigate the following multi-criteria problems: period/latency (Section 5.3), period/energy (Section 5.4) and period/latency/energy (Section 5.5). We discard the latency/energy combination since, as discussed above, the energy model holds only in combination with the period criterion.

When dealing with multiple criteria, our approach is to minimize one of them, given a threshold on the others. Actually, fixing the period or the latency means fixing a threshold on the period or latency of each application, thus providing a table of period or latency values. Equivalently, we minimize the value of Equation (6) with suitable coefficients. For the energy, only a bound on the global energy consumption is required. Note that all results apply to both the overlap and no-overlap models, and to all objective functions introduced in Section 4.4: more precisely, polynomial problems remain polynomial for arbitrary weights W_a in Equation (6), while NP-complete problems are already difficult with $W_a = 1$. All complexity results are summarized in Section 5.6.

5.1 Period minimization

We show that a greedy assignment solves the problem of finding a one-to-one mapping on communication homogeneous platforms, but the problem turns NP-complete with heterogeneous links between the processors. For interval mappings, we use an existing algorithm which finds the minimum period in a single application to build a new polynomial time algorithm that minimizes the global period of many applications on fully homogeneous platforms, giving the right number of processors to each application. The problem is NP-complete with heterogeneous processors, even for the case **special-app**.

5.1.1 One-to-one mappings

Theorem 1 *On communication homogeneous platforms, a one-to-one mapping that minimizes the period can be determined in polynomial time.*

Proof. The following proof is an adaptation of the algorithm described in [6], which finds the minimum period under the same hypothesis but for a single application. The main idea remains the same, since on communication homogeneous platforms the application that the stage belongs to does not matter for a one-to-one mapping.

The optimal period belongs to the set:

$$\mathcal{T} = \left\{ W_a \times \max\left(\frac{\delta_a^{k-1}}{b}, \frac{w_a^k}{s_u}, \frac{\delta_a^k}{b} \right) \right\}_{1 \leq a \leq A, 1 \leq k \leq n_a, 1 \leq u \leq p},$$

because it is equal to the product of W_a by the cycle-time of some processor P_u, running in its fastest mode s_u, and executing one of the N stages, S_a^k. First we compute the set $\mathcal{T}$ and we sort its elements into an array $\mathcal{T}_A$. Then, we perform a binary search on the array $\mathcal{T}_A$ to find the optimal period, testing at each step whether the current element T is a feasible value. To do so, we use the greedy assignment procedure of Algorithm 1. Initially, the current element T is the median of $\mathcal{T}_A$. If the greedy assignment procedure returns "failure", we increase the period by jumping to the median of the elements of $\mathcal{T}_A$ which are larger

Algorithm 1 Greedy-Assignment(T)

Work with fastest N processors, numbered P_1 to P_N, where $s_1 \leq s_2 \leq \cdots \leq s_N$
Mark all stages $\mathcal{S}_1$ to $\mathcal{S}_N$ as free
for $u = 1$ to N **do**
  Pick up any free stage $\mathcal{S}_a^k$ such that:
    $W_a \times \max(\frac{\delta_a^{k-1}}{b_a}, \frac{w_a^k}{s_u}, \frac{\delta_a^k}{b_a}) \leq T$
  Assign $\mathcal{S}_a^k$ to P_u
  Mark $\mathcal{S}_a^k$ as already assigned
  if no stage found **then**
    return "failure"
  end if
end for
return "success"

than T, and if it returns "success", we jump to the median of the elements of $\mathcal{T}_A$ which are smaller than T. The algorithm terminates in $\lceil \log \mathcal{T} \rceil$ iterations.

Note that $|\mathcal{T}| \leq N \times p$ (N stages and p processors), hence the total computation time is $O((N \times p + cost_{GA}) \log(N \times p))$, where $cost_{GA}$ is the cost of the greedy assignment procedure.

We now describe the greedy assignment algorithm for a prescribed value T of the achievable period. Recall that there are N stages to map onto $p \geq N$ processors in a one-to-one fashion. Also, we target communication homogeneous platforms with different-speed processors ($s_u \neq s_v$), with different-capacity links between the applications, but with links of same capacities within an application. First we retain only the fastest N processors, which we rename $P_1, P_2, \ldots, P_N$ such that $s_1 \leq s_2 \leq \cdots \leq s_N$. Then we consider the processors in the order P_1 to P_N, i.e., from the slowest to the fastest, and greedily assign them any free (not already assigned) task that they can process within the period.

The proof that the greedy procedure returns a solution if and only if there exists a solution of period T is done by a simple exchange argument. Indeed, consider a valid one-to-one assignment of period T, denoted $\mathcal{A}$, and assume that it has assigned stage $\mathcal{S}_{a_1}^{k_1}$ to P_1. Note first that the greedy procedure will indeed find a stage to assign to P_1 and cannot fail, since $\mathcal{S}_{a_1}^{k_1}$ can be chosen. If the choice of the greedy procedure is actually $\mathcal{S}_{a_1}^{k_1}$, we proceed by induction with P_2. If the greedy procedure has selected another stage $\mathcal{S}_{a_2}^{k_2}$ for P_1, we find which processor, say P_u, has been assigned this stage in the valid assignment $\mathcal{A}$. Then we exchange the assignments of P_1 and P_u in $\mathcal{A}$. As P_u is faster than P_1, which could process $\mathcal{S}_{a_1}^{k_1}$ in time in the assignment $\mathcal{A}$, P_u can process $\mathcal{S}_{a_1}^{k_1}$ in time too.

As $\mathcal{S}_{a_2}^{k_2}$ has been mapped on P_1 by the greedy procedure, P_1 can process $\mathcal{S}_{a_2}^{k_2}$ in time. So the exchange is valid, we can consider the new assignment which is valid and which did the same assignment on P_1 than the greedy procedure. The proof proceeds by induction with P_2 as before.

The complexity of the greedy assignment procedure is $cost_{GA} = O(N^2)$, because of the two loops over processors and stages. Altogether, since $N \leq p$, the cost of Algorithm 1 can be neglected, and the complexity of the whole

algorithm is $O((N \times p)\log(N \times p))$, which is indeed polynomial in the problem size.

In addition, note that this algorithm works with the no-overlap communication model, by replacing $W_a \times \max(\frac{\delta_a^{k-1}}{b_a}, \frac{w_a^k}{s_u}, \frac{\delta_a^k}{b_a}) \leq T$ by $W_a \times (\frac{\delta_a^{k-1}}{b_a} + \frac{w_a^k}{s_u} + \frac{\delta_a^k}{b_a}) \leq T$. ■

Theorem 2 *On fully heterogeneous platforms, the problem of finding a one-to-one mapping that minimizes the period is NP-complete.*

Proof. As the problem was already NP-complete with one single application [6], it remains NP-complete with concurrent applications. ■

5.1.2 Interval mappings

Theorem 3 *On fully homogeneous platforms, an interval mapping that minimizes the period can be determined in polynomial time.*

Proof. A polynomial algorithm has already been found to exhibit the minimal period with one application, under a communication model without overlap [6], and it can easily be extended to the overlap model, so the following proof is valid for both models. We exhibit an algorithm (see Algorithm 2) which finds an optimal interval mapping for concurrent applications, thanks to the previous polynomial algorithm for a single application, and we show its validity.

Algorithm 2

```
Assign all stages of each application to one processor
Compute the period of all applications
for a = (p − A) to p do
    Find an application a' such that W_a' × T_a' is maximum
    Add one processor to this application
    Compute the new period T_a' of this application
end for
```

First, here are some notations:

- $(k_{a,i}^u)$ is a A-tuple which represents the processor distribution among the applications at step i of Algorithm 2.
- $(k_{a,i}^o)$ is an optimal processor distribution with i processors.
- $T_a(n)$ is the period of the application numbered a, where n is the number of processors the application a is assigned to.
- $T(d) = \max_{a \in \{1,\ldots,A\}} W_a \times T_a(d_a)$, where d is a A-tuple.

Let us prove now the optimality of Algorithm 2.

- $(k_{a,A}^u)$ is the best distribution with A processors, because it is the only one.
- Let us assume that $(k_{a,i}^u)$ is optimal with i processors. We want $(k_{a,i+1}^u)$ to be an optimal distribution with $i+1$ processors.
  - Either: $\exists a, k_{a,i+1}^o < k_{a,i}^u$
    In this case, by construction,

$$\exists i' < i, T((k_{a,i'}^u)) = W_a \times T_a(k_{a,i'}^u) = W_a \times T_a(k_{a,i+1}^o)$$

Now, because every T_a and $x \mapsto W_a \times x$ are non-decreasing, $T((k^u_{a,i+1})) \leq T((k^u_{a,i})) \leq T((k^u_{a,i'}))$, and by definition $W_a \times T_a(k^o_{a,i+1}) \leq T((k^o_{a,i+1}))$. Finally, $T((k^u_{a,i+1})) \leq T((k^o_{a,i+1}))$.

– Or: $\exists! a, k^o_{a,i+1} = k^u_{a,i} + 1$
  * either: $k^u_{a,i+1} = k^u_{a,i} + 1$ and we are done,
  * or: $\exists a' \neq a, k^u_{a',i+1} = k^u_{a',i} + 1$
    In this case, by construction, $T((k^u_{a,i})) = f_{a'}(T_{a'}(k^u_{a',i})) = f_{a'}(T_{a'}(k^o_{a',i+1}))$ because $k^u_{a',i} = k^o_{a',i+1}$. Thus $T((k^u_{a,i})) \leq T((k^o_{a,i+1}))$. Finally, $T((k^u_{a,i+1})) \leq T((k^o_{a,i+1}))$.

Overall we have shown that $(k^u_{a,i+1})$ was as good as $(k^o_{a,i+1})$.

- By induction, the algorithm finds an optimal solution to map A applications onto p processors.

The complexity of computing the period of application a with $q \leq p$ processors, keeping the intermediate result with $q - 1$ processors, is bounded by $O((n_a)^3 q)$ [6]. Let $n_{max} = \max_{a \in \{1,\ldots,A\}} n_a$. Since we perform at most p steps in the algorithm, and $q \leq p$, the complexity of Algorithm 2 is bounded by $O(n^3_{max} p^2)$, which is indeed polynomial in the problem size. ■

Theorem 4 *On communication homogeneous platforms, the problem of finding an interval mapping that minimizes the period is NP-complete.*

Proof. As the problem was already NP-complete with one single application [6], it remains NP-complete with concurrent applications. ■

The case **special-app** is more interesting, because a polynomial algorithm exists to find an interval mapping which minimizes the period of one single application [7]; however, the problem becomes NP-complete with several applications.

Theorem 5 *With several applications, heterogeneous processors, homogeneous pipelines without communication, the problem of finding an interval mapping which minimizes respectively* $\max_{a \in \{1,\ldots,A\}} T_a$, $\max_{a \in \{1,\ldots,A\}} W_a \times T_a$, *or* $\max_{a \in \{1,\ldots,A\}} T_a / T^*_a$, *are NP-complete (in the strong sense).*

Proof. First we focus on the first problem, i.e., minimizing $\max_{a \in \{1,\ldots,A\}} T_a$.

We consider the associated decision problem: given a period T, is there a mapping of period less than T? The problem is obviously in NP: given a period and a mapping, it is easy to check in polynomial time that it is valid by computing its period.

To establish the completeness, we use a reduction from 3-PARTITION [14]. We consider an instance $\mathcal{I}_1$ of 3-PARTITION: given an integer B and $3m$ positive integers $a_1, a_2, \ldots, a_{3m}$ such that for all $i \in \{1, \ldots, 3m\}$, $B/4 < a_i < B/2$ and with $\sum_{i=1}^{m} a_i = mB$, does there exist a partition $I_1, \ldots, I_m$ of $\{1, \ldots, 3m\}$ such that for all $j \in \{1, \ldots, m\}$, $|I_j| = 3$ and $\sum_{i \in I_j} a_i = B$?

As 3-PARTITION is NP-complete in the strong sense, we can encode the $3m$ numbers in unary, and assume that the size of $\mathcal{I}_1$ is $O(mB)$.

We build an instance $\mathcal{I}_2$ of our problem with m identical applications such that each application is composed of B stages, with $w = 1$, and $p = 3m$ processors with speeds a_j for each $j \in \{1, \ldots, 3m\}$. We ask whether it is possible to realize a period of 1. Clearly, the size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$

(coded in unary). We now show that instance $\mathcal{I}_1$ has a solution if and only if instance $\mathcal{I}_2$ does.

Suppose first that $\mathcal{I}_1$ has a solution. Let $I_j = \{a'_{1,j}, a'_{2,j}, a'_{3,j}\}$, for $j \in \{1, \ldots, m\}$. For each $j \in \{1, \ldots, m\}$, we assign the $a'_{1,j}$ first consecutive stages of the application j to the processor of speed $a'_{1,j}$, the $a'_{2,j}$ next stages to the processor of speed $a'_{2,j}$, and the $a'_{3,j}$ remaining stages to the processor of speed $a'_{3,j}$. As the period of every processor is clearly equal to 1, the period is 1.

Suppose now that $\mathcal{I}_2$ has a solution. As the sum of all computation times is equal to the sum of all processor speeds, and a processor cannot be assigned stages of two different applications, for each application, the sum of its computation times is equal to the sum of the speed of processors which are assigned a stage of this application. Now, for all $i \in \{1, \ldots, 3m\}$, $B/4 < a_i < B/2$, so there are exactly three processors involved in the processing of each application. We can derive easily a solution to $\mathcal{I}_1$ (set I_j corresponding to processors of application j).

As there is no communication, this proof is valid for both communication models.

For the second problem, we follow the previous proof, but we assume now that, for each $a \in \{1, \ldots, A\}$, for $k \in \{1, \ldots, m\}$, $w_a^k = 1/W_a$. Then we scale each application: each w_a^k is multiplied by W_a so that the new period T'_a of the application a will be $W_a T_a$. We are now in the case of the previous proof.

Finally, for the third problem, we build the same instance as the one of the first proof. As the pipeline applications are all similar, the period of those applications when they are alone on the platform are all the same. We finally just have to minimize $\max_{a \in \{1,\ldots,A\}} T_a$. ■

5.2 Latency minimization

We show that finding a one-to-one mapping which minimizes the latency is NP-complete as soon as the processors do not have the same speed thanks to a reduction from 3-PARTITION. However we write a greedy algorithm that finds the optimal interval mapping on communication homogeneous platforms. The problem is still NP-complete on fully heterogeneous platforms for interval mappings.

Note that latency expression does not depend on the communication model, thus the results of this section are valid for the overlap and no-overlap models.

5.2.1 One-to-one mappings

Theorem 6 *The problem of finding the one-to-one mapping which minimizes the latency on fully homogeneous platforms is polynomial.*

Proof. As all mappings are equivalent, the theorem is true. ■

The case **special-app** is more interesting, because a polynomial algorithm exists to find a one-to-one mapping which minimizes the latency of one single application [8]; however, the problem becomes NP-complete with several concurrent applications.

Theorem 7 *With several applications, heterogeneous processors, homogeneous pipelines without communication, the problem of finding the optimal one-to-one mapping which minimizes respectively* $\max_{a\in\{1,\ldots,A\}} L_a$, $\max_{a\in\{1,\ldots,A\}} W_a \times L_a$, *or* $\max_{a\in\{1,\ldots,A\}} L_a/L_a^*$, *are NP-complete (in the strong sense).*

Proof. First we focus on the first problem, i.e., minimizing $\max_{a\in\{1,\ldots,A\}} L_a$.

We consider the associated decision problem: given a latency L, is there a mapping of latency less than L? The problem is obviously in NP: given a latency and a mapping, it is easy to check in polynomial time that it is valid by computing its latency.

To establish the completeness, we use a reduction from 3-PARTITION. We consider an instance $\mathcal{I}_1$ of 3-PARTITION: given an integer B and $3m$ positive integers $a_1, a_2, \ldots, a_{3m}$ such that for all $i \in \{1,\ldots,3m\}$, $B/4 < a_i < B/2$ and with $\sum_{i=1}^{m} a_i = mB$, does there exists a partition $I_1, \ldots, I_m$ of $\{1,\ldots,3m\}$ such that for all $j \in \{1,\ldots,m\}$, $|I_j| = 3$ and $\sum_{i\in I_j} a_i = B$?

We build an instance $\mathcal{I}_2$ of our problem with m identical applications, each one composed of 3 stages with $w = 1$, and $p = 3m$ processors with speeds $1/a_j$ for $j \in \{1,\ldots,3m\}$. We ask whether it is possible to realize a global latency of B. Clearly, the size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$. We now show that instance $\mathcal{I}_1$ has a solution if and only if instance $\mathcal{I}_2$ does.

Suppose first that $\mathcal{I}_1$ has a solution. Let, for each $j \in \{1,\ldots,m\}$, $I_j = \{a'_{1,j}, a'_{2,j}, a'_{3,j}\}$. For each $j \in \{1,\ldots,m\}$, for $i \in \{1,2,3\}$ we assign the i^{th} stage of the application j to the processor whose speed is equal to $1/a'_{i,j}$. The global latency is clearly B.

Suppose now that $\mathcal{I}_2$ has a solution. There exists a partition $I_1, \ldots, I_m$ of $\{1,\ldots,3m\}$ such that for all $j \in \{1,\ldots,m\}$, $|I_j| = 3$ and $\sum_{i\in I_j} a_i \leq B$. Since $\sum_{i=1}^{m} a_i = mB$, we have, $\forall j \in \{1,\ldots,m\}$, $\sum_{i\in I_j} a_i = B$. We conclude that $\mathcal{I}_1$ has a solution.

For the second problem, the proof is the same as the previous one, but we have now $w_a^1 = w_a^2 = w_a^3 = 1/W_a$.

For the third problem, the proof is similar to the first one, but we ask now whether it is possible to realize a global latency of $K \times B$, where K is the sum of the three biggest a_i. All applications have indeed the same latency when they are alone on the platform, and this latency is K. Instead of minimizing $\max_{a\in\{1,\ldots,A\}} \frac{L_a}{L_a^*}$, we minimize $\max_{a\in\{1,\ldots,A\}} \frac{L_a}{K}$ so we minimize $\max_{a\in\{1,\ldots,A\}} L_a$. ■

5.2.2 Interval mappings

Theorem 8 *On communication homogeneous platforms, the optimal interval mapping which minimizes the latency can be determined in polynomial time.*

Proof. First, note that with a single application, the optimal mapping is obtained by mapping the whole application onto one processor. Indeed, if two distinct processors were enrolled in the computation, mapping the entire application onto the fastest processor would reduce the computation time and remove the communication cost. Therefore, with several concurrent applications, we keep the A fastest processors and map the applications onto those processors in

a one-to-one fashion. The greedy procedure written for the period minimization problem with one-to-one mapping can be reused.

The optimal latency belongs to the set:

$$\mathcal{L} = \left\{ W_a \times \left(\frac{\delta_a^0}{b} + \frac{\sum_{k=1}^{n_a} w_a^k}{s_u} + \frac{\delta_a^{n_a}}{b} \right) \right\}_{1 \leq a \leq A,\ 1 \leq u \leq p}.$$

Since $|\mathcal{L}| = Ap$, the complexity of the algorithm is $O((Ap + A^2)\log(Ap))$, and it can be simplified in $O(Ap \log(Ap))$. ■

Theorem 9 *On fully heterogeneous platforms, the problem of finding an optimal interval mapping, that minimizes the latency, is NP-complete.*

Proof. As the problem of finding the interval mapping, which minimizes the latency on fully heterogeneous platforms, was already NP-complete with one single application [8], it remains NP-complete with several concurrent applications. ■

5.3 Period/latency minimization

In this section again, we are not concerned with energy minimization issues, so, similarly to results of Sections 5.1 and 5.2, all processors can be run systematically at their highest speed. Therefore, on fully homogeneous platforms, all one-to-one mappings are identical, and it is straightforward to minimize the latency for a given period, or the converse.

However, for interval mappings, we must decide where to split applications into intervals, and we provide a dynamic programming algorithm which solves both variants of the problem with a single application. When considering multiple applications, we need to run the dynamic programming algorithm once per application with the corresponding period (resp. latency) threshold, and the minimum latency (resp. period) that can then be achieved is the maximum over all applications.

Theorem 10 *With one application, on fully homogeneous platforms, the optimal interval mapping which minimizes the latency for a bounded period, or the period for a bounded latency, can be determined in polynomial time.*

Proof. We denote by n the number of stages, s the speed of every processor and b their bandwidth.

We exhibit a dynamic programming algorithm which computes the optimal mapping that minimizes the latency for a given period. We compute recursively the values of $(L, T)(i, q)$, which are the optimal latency and period that can be achieved by any interval-based mapping of stages $\mathcal{S}^1$ to $\mathcal{S}^i$ using exactly q processors. The recurrence relation can be expressed as:

$$(L,T)(i,q) = \min_{1 \leq j < i} \left\{ \begin{array}{ll} \left(L(j, q-1) + \frac{\sum_{k=j+1}^{i} w^k}{s} + \frac{\delta^i}{b}, \right. \\ \max \quad \left. \left(T(j, q-1), \max(\frac{\delta^j}{b}, \frac{\sum_{k=j+1}^{i} w^k}{s}, \frac{\delta^i}{b}) \right) \right) \end{array} \right\}.$$

This relation holds for all $i > 1$ and $q > 1$. The function "min" keeps the brace such that the period is not greater than the given period and the latency is minimum. If such a brace does not exist, it returns $(+\infty, +\infty)$.

The initialization relations are:

- If there is only one processor, we map the whole interval onto this processor. For each $i \in \{1, \ldots, n\}$:

$$(L,T)(i,1) = \left(\frac{\delta^0}{b} + \frac{\sum_{k=1}^{i} w^k}{s} + \frac{\delta^i}{b}, \max\left(\frac{\delta^0}{b}, \frac{\sum_{k=1}^{i} w^k}{s}, \frac{\delta^i}{b}\right)\right)$$

- If $q > 1$ (too many processors for one stage):

$$(L,T)(1,q) = (+\infty, +\infty)$$

Finally we aim at computing:

$$\min_{q \in \{1,\ldots,p\}} (L,T)(n,q) \ .$$

This dynamic programming algorithm solves the problem of finding a mapping, which minimizes the latency for a given period, with a complexity in $O(n^2 p)$.

For the converse problem of finding a mapping which minimizes the period for a given latency, we use a binary search. The minimum period belongs to the set:

$$\mathcal{T} = \left\{ \frac{\sum_{k=i}^{j} w^k}{s} \right\}_{1 \leq i \leq j \leq n} \bigcup \left\{ \frac{\delta^i}{b} \right\}_{0 \leq i \leq n}$$

Moreover, if a mapping realizes a period T and a latency L, then it realizes a period $T_2 > T$ and a latency $L_2 = L$. We conclude that the algorithm which minimizes the latency for a given period T_{lim} will find a bigger latency than the one which minimizes the latency for a given period $T_{lim}^2 > T_{lim}$. We can thus minimize the period for a given latency thanks to a binary search on the period and some calls to the previous algorithm, which minimizes the latency for a given period.

Since $|\mathcal{T}| = \frac{n(n+1)}{2} + n$, the complexity of this problem is $O((n^2 + n^2 p)\log(n))$, i.e. $O(n^2 p \log(n))$.

The proof of this theorem under the no-overlap communication model is very similar: all we have to do is to replace $\max(\frac{\delta^j}{b}, \frac{\sum_{k=j+1}^{i} w^k}{s}, \frac{\delta^i}{b})$ by $\frac{\delta^j}{b} + \frac{\sum_{k=j+1}^{i} w^k}{s} + \frac{\delta^i}{b}$ in the recurrence relation, $\max\left(\frac{\delta^0}{b}, \frac{\sum_{k=1}^{i} w^k}{s}, \frac{\delta^i}{b}\right)$ by $\frac{\delta^0}{b} + \frac{\sum_{k=1}^{i} w^k}{s} + \frac{\delta^i}{b}$ in the first initialization relation, and the previous $\mathcal{T}$ by

$$\mathcal{T} = \left\{ \frac{\delta^{i-1}}{b} + \frac{\sum_{k=i}^{j} w^k}{s} + \frac{\delta^j}{b} \right\}_{1 \leq i \leq j \leq n} . \ \blacksquare$$

Theorem 11 *With several applications, on fully homogeneous platforms, the optimal interval mapping which minimizes the latency $L = \max_{a \in \{1,\ldots,A\}} W_a \times L_a$ for a bounded period by application, or the period $T = \max_{a \in \{1,\ldots,A\}} W_a \times T_a$ for a bounded latency by application can be determined in polynomial time.*

Proof. For several applications, we can reuse the structure of Algorithm 2, but instead of computing the period, we compute both period and latency, thanks to one of the previous algorithms for one single application (dynamic programming algorithm if we minimize the global latency for given periods, and binary search combined with dynamic programming algorithm if we minimize

the global period for given latencies, see proof of Theorem 10). While there are some processors which are not yet allocated, we add one processor to any application which maximizes the criterion we want to minimize (if the bound on the other criterion is exceeded, the first criterion is set to $+\infty$, according to the single application algorithm).

Since there is a total of p calls to the single-application algorithms, and a total of N application stages, the complexity is in $O((Np)^2 \log(N))$ for the period minimization with a bounded latency, and in $O((Np)^2)$ for the latency minimization with a bounded period. ■

When moving to a platform with heterogeneous processors, even if the application is homogeneous with no communication (case **special-app**), the problem of finding a one-to-one or interval mapping that solves the bi-criteria period/latency problem is NP-complete. This result is a direct consequence of the NP-completeness of the mono-criterion cases, see Sections 5.1 and 5.2.

Theorem 12 *With heterogeneous processors and homogeneous pipelines, without communication, the problem of finding an interval or one-to-one mapping, that solves the bi-criteria period/latency problem, is NP-complete.*

Proof. The problem of minimizing the latency with a one-to-one mapping is NP-complete, so finding a one-to-one mapping that minimizes the latency for a given array of period is NP-complete too.

The problem of minimizing the period with an interval mapping is NP-complete, so finding an interval mapping that minimizes the period for a fixed latency by application is NP-complete too. ■

5.4 Period/energy minimization

We first provide results for one-to-one mappings, and then discuss interval mappings. For fully heterogeneous platforms, the problem is NP-hard because the period minimization problem already is NP-hard on such platforms. The interesting result is the following:

Theorem 13 *On communication homogeneous platforms, a one-to-one mapping which minimizes the energy consumption while enforcing a given period for each application can be determined in polynomial time.*

Proof. We build a bipartite graph $G = (U, V, E)$, and prove that the problem amounts to finding a minimum weighted matching in this graph. U is the processor set, and V the stage set. For each processor and each stage, the weight of the edge between the two vertices is set to $+\infty$ if the processor cannot execute the stage within the period, and else it is the energy consumed by the processor when it is running in the smallest mode allowing to execute the stage within the period. Finding a minimum weighted matching gives us the minimum power consumption, in polynomial time $O\left((N+p)^{\frac{3}{2}}\right)$. ■

For interval mappings, first note that the problem becomes NP-complete as soon as we consider different speed processors, because of the NP-completeness of the period minimization problem for such platforms. Thus we focus on fully homogeneous platforms.

Theorem 14 *On fully homogeneous platforms, an interval mapping which minimizes the energy consumption while enforcing a given period for each application can be determined in polynomial time.*

Proof. We first exhibit a dynamic programming algorithm that returns the optimal energy consumption for a single application, when using exactly k processors to compute the application. This algorithm is fixing the processor speeds so as to minimize the energy. Then, the multiple application case can be solved using another dynamic programming algorithm, which decides how many processors should be allocated to each application.

For a single application $a \in \{1, \ldots, A\}$, and a processor number $q \in \{1, \ldots, p\}$, we compute E_a^q, the minimum energy consumed for the application a using at most q processors. We recursively compute the value $E(i, j, k)$, which is the optimal energy consumption that can be achieved by any interval-based mapping of stages $\mathcal{S}_a^i$ to $\mathcal{S}_a^j$ using exactly k processors. The goal is to determine $E_a^q = \min_{k \in \{1,\ldots,q\}} E(1, n_a, k)$. The recurrence relation can be expressed as:

$$E(i, j, k) = \min_{i \leq \ell \leq j-1} (E(i, \ell, k-1) + E(\ell+1, j, 1))$$

with the initialization:

- $E(i, i, r) = +\infty$ if $r > 1$
- Defining

$$\mathcal{F}_i^j = \left\{ E_{dyn}(s_\ell) + E_{stat} \;\middle|\; \max\left(\frac{\delta^{i-1}}{b}, \frac{\sum_{k=i}^{j} w^k}{s_\ell}, \frac{\delta^j}{b}\right) \leq T \right\}_{1 \leq \ell \leq m}$$

we have:

$$E(i, j, 1) = \begin{cases} \min \mathcal{F}_i^j & \text{if } \mathcal{F}_i^j \neq \varnothing \\ +\infty & \text{otherwise} \end{cases}$$

Here, m is the number of speed modes, and T is the period bound for the application a. The complexity of this dynamic programming algorithm is bounded by $O(n_a^2(p+m))$.

Note that for the no-overlap model, we simply replace $\max\left(\frac{\delta^{i-1}}{b}, \frac{\sum_{k=i}^{j} w^k}{s_\ell}, \frac{\delta^j}{b}\right)$ by $\frac{\delta^{i-1}}{b} + \frac{\sum_{k=i}^{j} w^k}{s_\ell} + \frac{\delta^j}{b}$ in the definition of $\mathcal{F}_i^j$. Note also that $E_a^k = +\infty$ if the algorithm fails to match the period T.

For several applications, let $E(a, k)$ the minimum energy consumed by k processors on the first applications $1, \ldots, a$, so we are looking for $E(A, p)$. This energy can be computed recursively, thanks to the recurrence relation:

$$\forall k \in \{1, \ldots, p\}, \forall a \in \{2, \ldots, A\}, E(a, k) = \min_{q \in \{0,\ldots,k-1\}} (E_a^q + E(a-1, k-q))$$

and the initialization: $\forall k \in \{1, \ldots, p\},\ E(1, k) = E_1^k$.

The overall complexity is $O(AN^3p^2)$. ■

5.5 Period/latency/energy minimization

When mixing the three criteria, the problem becomes NP-hard even for fully homogeneous platforms, no communication, and a single application. The combinatorial nature of the problem comes from the fact that even if processors are identical, they are multi-modal and each of them may run at a different speed.

Theorem 15 *On fully homogeneous platforms, with a single application and without any communication cost, finding a one-to-one mapping that solves the tri-criteria problem is NP-hard.*

Proof. We consider the associated decision problem: given a period T, a latency L and an energy E, does there exist a one-to-one mapping of period less than T, latency less than L and energy less than E?

The problem is obviously in NP: given a period, a latency, an energy and a mapping, it is easy to check in polynomial time that the mapping is valid.

To establish the completeness, we use a reduction from 2-PARTITION [14]. We consider an instance $\mathcal{I}_1$ of 2-PARTITION: given n strictly positive integers $a_1, a_2, \ldots, a_n$, does there exists a subset I of $\{1, \ldots, n\}$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$? Let $S = \sum_{i=1}^{n} a_i$. Let $K = \alpha \times S + 2$, where α is the exponent used in the computation of the energy (see Section 4.5).

We build an instance $\mathcal{I}_2$ of our problem with n identical processors, each with $m = 2n + 1$ modes such that:

$$\forall i \in \{1, \ldots, n\} \quad \begin{cases} s_{2i-1} = K^i \\ s_{2i} = K^i + \frac{a_i X}{K^{i(\alpha-1)}} \end{cases}$$

and a pipelined application composed of n stages, with computation costs $w_i = K^{i(\alpha+1)}$.

Intuitively, the idea is to choose K such that (i) stage weights are far enough from one another; and (ii) there is a gap between (s_{2i-1}, s_{2i}) and (s_{2j-1}, s_{2j}). Then the mapping will use exactly one component of every pair (s_{2i-1}, s_{2i}).

We claim that for each $j \in \{2, \ldots, n\}$, we have $K^{j\alpha} > \sum_{i=1}^{j-1} K^{i\alpha} + \alpha\left(\frac{S}{2} - \frac{1}{2}\right)$ and $K^{j\alpha+1} > \sum_{i=1}^{j} K^{i\alpha} + \left(K^{1-\alpha} \times a_{j-1} + 1 - \frac{S}{2}\right)$.

To prove the claim, let $j \in \{2, \ldots, n\}$. On the one side,

$$\begin{aligned} \textstyle\sum_{i=1}^{j-1} K^{i\alpha} + \alpha\left(\frac{S}{2} - \frac{1}{2}\right) &< \textstyle\sum_{i=1}^{j-1} K^{i\alpha} + \alpha S \\ &< (j-1)K^{(j-1)\alpha} + K \\ &< jK^{(j-1)\alpha} < K^{j\alpha} . \end{aligned}$$

On the other side,

$$\begin{aligned} &\textstyle\sum_{i=1}^{j} K^{i\alpha} + \left(K^{1-\alpha} \times a_{j-1} + 1 - \frac{S}{2}\right) \\ &\quad < \textstyle\sum_{i=1}^{j} K^{i\alpha} + K^{1-\alpha} \times K \\ &\quad < jK^{j\alpha} + K^{2-\alpha} < (j+1)K^{j\alpha} < K^{j\alpha+1} . \end{aligned}$$

We deduce that for each $j \in \{2, \ldots, n\}$ and each $0 < X < 1$, $K^{j\alpha} > \sum_{i=1}^{j-1} K^{i\alpha} + \alpha X\left(\frac{S}{2} - \frac{1}{2}\right)$ and $K^{j\alpha+1} > \sum_{i=1}^{j} K^{i\alpha} + X\left(K^{1-\alpha} \times a_{j-1} + 1 - \frac{S}{2}\right)$.

For all $i \in \{1, \ldots, n\}$, if we choose speed s_{2i} instead of speed s_{2i-1}, the additional energy is:

$$\begin{aligned} s_{2i}^{\alpha} - s_{2i-1}^{\alpha} &= (K^i + \frac{a_i X}{K^{i(\alpha-1)}})^{\alpha} - K^{i\alpha} \\ &= K^{i\alpha}(1 + \alpha\frac{a_i X}{K^{i\alpha}} + o(X)) - K^{i\alpha} \\ &= \alpha a_i X + f_i^E(X) , \end{aligned}$$

where $f_i^E(X) \underset{x \to 0}{=} o(X)$.

In the same way, for each $i \in \{1,\dots,n\}$, the difference in latency when using speed s_{2i} instead of speed s_{2i-1} to execute stage $\mathcal{S}_i$ is:

$$\begin{aligned}\frac{w_i}{s_{2i-1}} - \frac{w_i}{s_{2i}} &= \frac{K^{i(\alpha+1)}}{K^i} - \frac{K^{i(\alpha+1)}}{K^i + \frac{a_i X}{K^{i(\alpha-1)}}} \\ &= \frac{K^{i(\alpha+1)}}{K^i} - \frac{K^{i(\alpha+1)}}{K^i}\left(1 - \frac{a_i X}{K^{i\alpha}} + o(X)\right) \\ &= a_i X - f_i^L(X)\,,\end{aligned}$$

where $f_i^L(X) \underset{x\to 0}{=} o(X)$.

For all $i \in \{2,\dots,n\}$, the time to execute $\mathcal{S}_i$ at speed s_{2i-2} is:

$$\begin{aligned}\frac{w_i}{s_{2i-2}} &= \frac{K^{i(\alpha+1)}}{K^{i-1} + \frac{a_{i-1}X}{K^{(i-1)(\alpha-1)}}} \\ &= \frac{K^{i(\alpha+1)}}{K^{i-1}}\left(1 - \frac{a_{i-1}X}{K^{(i-1)\alpha}} + o(X)\right) \\ &= K^{i\alpha+1} - K^{1-\alpha} \times a_{i-1}X + f^{L_i}(X)\end{aligned}$$

So we choose $X < 1$ small enough, so that for each $i \in \{1,\dots,n\}$,

$$\begin{cases} |f_i^E(X)| < X \times \frac{\alpha}{2n} \\ |f_i^L(X)| < X \times \frac{1}{2n} \end{cases}$$

and for all $i \in \{2,\dots,n\}$, $|f^{L_i}(X)| < X \times \frac{1}{2}$.

We are now ready to choose the latency, energy and period bounds. Let E^* and L^* be the energy and latency obtained when $\mathcal{S}_i$ is executed at speed s_{2i-1} for all $i \in \{1,\dots,n\}$, $E^* = \sum_{i=1}^n s_{2i-1}^\alpha = \sum_{i=1}^n K^{i\alpha}$ and $L^* = \sum_{i=1}^n \frac{w_i}{s_{2i-1}} = E^*$. We ask whether it is possible to achieve an energy $E^o = E^* + \alpha X(S/2 + 1/2)$, a latency $L^o = L^* - X(S/2 - 1/2)$ and a period $T^o = L^o$.

Clearly, 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 $\mathcal{I}_2$ does.

Assume first that $\mathcal{I}_1$ has a solution. For each $i \in I$, stage $\mathcal{S}_i$ is executed at speed s_{2i}, and for each $i \in \{1,\dots,n\} \setminus I$, stage $\mathcal{S}_i$ is executed at speed s_{2i-1}. The mapping consumes an energy E and has a latency L, where:

$$\begin{aligned}E &= E^* + \textstyle\sum_{i\in I}\left(s_{2i}^\alpha - s_{2i-1}^\alpha\right) \\ &= E^* + \textstyle\sum_{i\in I}\left(\alpha a_i X + f_i^E(X)\right) \\ &\le E^* + \textstyle\sum_{i\in I}\left(\alpha a_i X + \frac{\alpha X}{2n}\right) \le E^* + \alpha X\left(\frac{S}{2} + \frac{1}{2}\right) \\ &\le E^o \\ L &= L^* - \textstyle\sum_{i\in I}\left(\frac{w_i}{s_{2i-1}} - \frac{w_i}{s_{2i}}\right) \\ &= L^* - \textstyle\sum_{i\in I}\left(a_i X - f_i^L(X)\right) \\ &\le L^* - \textstyle\sum_{i\in I}\left(a_i X - \frac{X}{2n}\right) \le L^* - X\left(\frac{S}{2} - \frac{1}{2}\right) \\ &\le L^o\end{aligned}$$

Because $T^o = L^o$, and because we fulfill the latency constraint, we fulfill the period constraint too. We conclude that $\mathcal{I}_2$ has a solution.

Suppose now that $\mathcal{I}_2$ has a solution. We first show that for each $i \in \{1,\dots,n\}$, stage $\mathcal{S}_i$ is executed at speed either s_{2i-1} or s_{2i}. Let $(\mathcal{P}_j)$ be the property: for each $i \in \{j,\dots,n\}$, there is a single processor running at speed s_{2i-1} or s_{2i}, and this processor is assigned stage $\mathcal{S}_i$. We first prove that $(\mathcal{P}_n)$ is true. On the one hand, if two processors were running at speed s_{2n-1} or s_{2n}, they would consume an energy
$E \geq 2s_{2n-1}^{\alpha} > K^{n\alpha} + \sum_{i=1}^{n-1} K^{i\alpha} + \alpha X \left(\frac{S}{2} + \frac{1}{2}\right) > E^o$.

On the other hand, if no processor was running at speed s_{2n-1} or s_{2n}, the latency would verify

$$\begin{aligned} L \geq \tfrac{w_n}{s_{2n-2}} &\geq K^{n\alpha+1} - K^{1-\alpha} \times a_{n-1}X + f^{L_i}(X) \\ &> \textstyle\sum_{i=1}^{n} K^{i\alpha} + X\left(K^{1-\alpha} \times a_{n-1} + 1 - \frac{S}{2}\right) \\ &\quad -K^{1-\alpha} \times a_{n-1}X + f^{L_i}(X) \\ &> \textstyle\sum_{i=1}^{n} K^{i\alpha} - X\left(\frac{S}{2} - \frac{1}{2}\right) + \left(\frac{X}{2} + f^{L_i}(X)\right) \\ &> L^o\,. \end{aligned}$$

We conclude that $(\mathcal{P}_n)$ is true. We now proceed by induction. If for some $j \in \{3,\dots,n\}$, $(\mathcal{P}_j)$ is true, then we show that $(\mathcal{P}_{j-1})$ is true in a quite similar way. In the end, $(\mathcal{P}_2)$ is true (and the processor that is assigned stage $\mathcal{S}_1$ is running either at speed s_1, or at speed s_2). Let I the subset of $\{1,\dots,n\}$ such that the processor that is assigned the stage $\mathcal{S}_i$ is running at speed s_{2i}. Then for each $i \in \{1,\dots,n\}\backslash I$, the processor that is assigned stage $\mathcal{S}_i$ is running at speed s_{2i-1}. The consumed energy is $E = E^* + \sum_{i\in I}\left(\alpha a_i X + f_i^E(X)\right)$, and $E \leq E^o$, hence $\sum_{i\in I} a_i \leq \frac{S}{2} + \left(\frac{1}{2} - \frac{\sum_{i\in I} f_i^E(X)}{\alpha X}\right)$. Therefore $\sum_{i\in I} a_i < \frac{S}{2} + \left(\frac{1}{2} + \frac{1}{2}\right)$. Since the a_i are integers, we conclude that $\sum_{i\in I} a_i \leq \frac{S}{2}$.

The achieved latency is $L = L^* - \sum_{i\in I}\left(a_i X - f_i^L(X)\right)$, and $L \leq L^o$, hence $\sum_{i\in I} a_i \geq \frac{S}{2} - \left(\frac{1}{2} - \frac{\sum_{i\in I} f_i^L(X)}{X}\right)$. Since $\frac{\sum_{i\in I} f_i^L(X)}{X} \leq \frac{1}{2}$, we get $\sum_{i\in I} a_i \geq \frac{S}{2}$.

Finally, $\sum_{i\in I} a_i = \frac{S}{2}$ and $\mathcal{I}_1$ has a solution, which concludes the proof. ■

Theorem 16 *On fully homogeneous platforms, with a single application and without any communication cost, finding an interval mapping that solves the tri-criteria problem is NP-hard.*

Proof. We only give the sketch of the completeness proof, which reuses the proof of Theorem 15. To construct the instance $\mathcal{I}_2$, we insert big stages between the previous stages. We add a big speed to the processor modes, adjusted to allow the execution of exactly one big stage during the period. More formally, we build a pipeline composed of $2n-1$ stages, such that $\forall i \in \{1,\dots,n\}, w_{2i-1} = K^{i(\alpha+1)}$ and $\forall i \in \{1,\dots,n-1\}, w_{2i} = K^{(n+1)(\alpha+1)}$ We use $2n-1$ identical processors, that can run $2n+1$ modes, such that $\forall i \in \{1,\dots,n\}, s_{2i-1} = K^i$ and $s_{2i} = K^i + \frac{a_i X}{K^{i\alpha}}$. We also let $s_{2n+1} = K^{n+1}$.

We search for an interval mapping, whose energy does not exceed $E^o = (n-1)K^{(n+1)\alpha} + E^* + \alpha X(S/2+1/2)$, whose latency does not exceed $L^o = (n-1)K^{(n+1)\alpha} + L^* - X(S/2-1/2)$, and whose period does not exceed $T^o = K^{(n+1)\alpha}$. If the instance $\mathcal{I}_1$ of 2-PARTITION has a solution, we proceed like in the previous proof, and map every big stage onto a processor that is running in its highest mode. All constraints are fulfilled.

If the instance $\mathcal{I}_2$ has a solution, we have to run processors that are assigned a big stage in their highest mode. Moreover, these processors cannot be assigned

	proc-hom **com-hom**	**proc-het** **special-app**	**com-hom**	**com-het**
Per - *one-to-one*	polynomial (binary search)			NP-c.
Per - *interval*	poly (dyn. prog. + greedy)	NP-complete(*)	NP-complete	
Lat - *one-to-one*	polynomial	NP-complete(*)		NP-c.
Lat - *interval*	polynomial (binary search)			NP-c.
Per/Lat - *both*	polynomial	NP-complete		
Per/En - *one-to-one*	polynomial (minimum matching)			NP-c.
Per/En - *interval*	poly (dyn. prog.)	NP-complete		
Per/Lat/En - *both*	NP-complete			

Table 1: Complexity results with the PATH latency model.

other stages. All we have to do next is to find a one-to-one mapping of the unassigned stages, with the additional constraint that we cannot run the

remaining processors in their highest modes without exceeding the energy bound. We then conclude as in the proof of Theorem 15. ■

We conclude this section with some remarks on uni-modal processors. If we restrict to processors with a single execution mode, the problem becomes polynomial on fully homogeneous platforms, while it remains NP-hard otherwise (because of the NP-completeness of the period/latency problem which is also established with uni-modal processors). For one-to-one mappings, all mappings are equivalent on fully homogeneous platforms, but the algorithm is more sophisticated for interval mappings. We first write an algorithm which partitions the stages of a single application into intervals, for each of the three variants of the tri-criteria optimization problem, and then we use this algorithm for the multiple application problem. Details can be found in [4].

5.6 Summary of complexity results for the Path model

Table 1 summarizes all complexity results with the PATH latency model, for one-to-one and interval mappings without resource sharing.

For the mono-criterion problems, most NP-completeness proofs come from the single application problem which already was NP-hard, see [6, 8] for the proofs. The two special entries denoted with (*) are problem instances which could be solved in polynomial time for a single application, but becomes NP-hard with several ones. Remaining entries correspond to polynomial algorithms that were already existing for a single application and that have been extended for several ones.

For the bi-criteria problems, we provide new polynomial algorithms to minimize one of the criterion, given a bound on the other one. NP-completeness results are obtained from the mono-criterion complexity results.

Finally, the tri-criteria problem turns out to be NP-hard even for fully homogeneous platforms, no communication and a single application.

6 Complexity results with the Wavefront model

In the previous section, we have performed an exhaustive complexity study considering the PATH latency model, and hence restricting to mapping rules without resource sharing (one-to-one or interval mappings). We have provided new

polynomial algorithms for multiple applications and results of NP-completeness. However, when considering resource sharing and general mappings, we use the WAVEFRONT latency model, as explained in the framework (see Section 4.4).

In this section, we investigate the impact of this model on the complexity results. Since the latency definition is now closely related to the period definition, we consider only latency in combination with period. For the period/latency combination, we minimize the latency for a fixed period. For the tri-criteria problem, both period and latency are fixed, and we minimize the energy criterion.

Also, we do not restrict the study to one-to-one and interval mappings, but also discuss general mappings. It turns out that the period minimization problem is NP-hard for such mappings, even for fully homogeneous platforms, no communication and a single application. Therefore, all multi-criteria problems with general mappings are NP-hard.

All results are summarized in Table 2.

6.1 Period minimization

All complexity results for period minimization were already established in Section 5.1, except for general mappings. It turns out that the problem is NP-hard for general mappings, even for fully homogeneous platforms, no communication and a single application.

Theorem 17 *On fully homogeneous platforms with no communication, the problem of finding a general mapping that minimizes the period of a single application is NP-complete.*

Proof. The reduction is straightforward, with a reduction from 2-PARTITION [14]: the application consists of n stages and there are two identical processors. Stages must be partitioned in two sets of equal computational weight, which amounts to 2-partition the stages. ■

As a corollary, all multi-criteria problems are NP-hard for general mappings, since they all involve the period criterion (because of the energy and latency definitions).

6.2 Period/latency minimization

With heterogeneous processors and interval mappings, we already know that the period minimization problem is NP-hard, and therefore it remains NP-hard when combining it with the latency criterion. However, the result does not hold any longer for one-to-one mappings, while the bi-criteria problem was NP-hard

	proc-hom	**proc-het**		
	com-hom	**special-app**	**com-hom**	**com-het**
Per/* - *general*	NP-complete			
Per/Lat - *one-to-one*	polynomial			NP-complete
Per/Lat - *interval*	polynomial	NP-complete		
Per/Lat/En - *one-to-one*	polynomial			NP-complete
Per/Lat/En - *interval*	poly (dyn. prog.)	NP-complete		

Table 2: Complexity results with the WAVEFRONT latency model.

with the PATH latency model. Actually, with homogeneous communications, the latency of an application with n stages is always $(2n-1) \times T$, where T is the period of the application, and therefore the latency is minimized when the period is minimized. The bi-criteria problem amounts in this case to the period minimization problem, which is polynomial (binary search algorithm, see Section 5.1).

For homogeneous platforms, we propose below a polynomial algorithm for the period/latency/energy combination on homogeneous platforms and interval mappings. This algorithm can be used to solve the easier bi-criteria problem with no energy criterion.

6.3 Period/latency/energy minimization

As motivated earlier, we focus on the tri-criteria problem of minimizing energy under constraints on period ans latency. It turns out that this problem becomes polynomial for interval mappings without resource sharing on fully homogeneous platforms, while it was NP-complete with the classical definition of latency (see Theorem 16).

For one-to-one mappings, the problem is polynomial for **com-hom** platforms with different speed processors. Indeed, similarly to the period/latency problem, minimizing the latency is equivalent to minimizing the period for such mappings because of the WAVEFRONT latency model and the one-to-one mapping.

Theorem 18 *With the* WAVEFRONT *latency model, the tri-criteria problem is polynomial on fully homogeneous platforms for interval mappings without reuse.*

Proof. The optimal solution for interval mappings relies on an intricate nesting of two dynamic programming algorithms. The first one solves the problem with one single application: it recursively computes the optimal energy consumption that can be achieved by mapping one stage interval to exactly q processors. Then another dynamic programming algorithm finds the minimum energy consumption with several applications, recursively trying all possible distributions of processors to applications, and using the first algorithm to compute the optimal energy consumption for each application, given the number of processors allocated to this application.

For the single application problem, Let n be the number of stages of this application, P_{giv} be the given period, and L_{giv} be the given latency. First of all, note that the latency is given by $L = (2m-1) \times P_{\text{giv}}$, where m is the number of intervals. Therefore, we can compute a priori the maximum possible number of intervals in the mapping. Let $m^{\max}$ be this number; note that it cannot exceed n, the total number of stages, nor p, the number of processors: $m^{\max} = \min(n, p, \lfloor (\frac{L_{\text{giv}}}{P_{\text{giv}}} + 1)/2 \rfloor)$. If we use more intervals, the bound on the latency will be exceeded. Otherwise, we just have to check if the period constraint is fulfilled.

We exhibit a dynamic programming algorithm that returns the optimal energy consumption. We compute recursively the value $E(i, j, q)$, which is the optimal energy consumption that can be achieved by any interval-based mapping of stages $\mathcal{S}^i$ to $\mathcal{S}^j$ using exactly q processors. The goal is to determine $\min_{m \in \{1,\dots,m^{\max}\}} E(1, n, m)$. The recurrence relation can be expressed as:

$$E(i, j, q) = \min_{i \leq \ell \leq j-1} \left(E(i, \ell, q-1) + E(\ell+1, j, 1) \right),$$

with the initializations:

- $E(i,i,q) = +\infty$ if $q > 1$ (we cannot run one stage with many processors);
- $E(i,j,1) = \left\{ \begin{array}{l} \min \mathcal{F}^{i,j} \text{ if } \mathcal{F}^{i,j} \neq \varnothing \\ +\infty \text{ otherwise,} \end{array} \right.$
  where $\mathcal{F}^{i,j} = \left\{ \begin{array}{l} E_{dyn}(s) + E_{stat} \mid \\ \max\left(\frac{\delta^{i-1}}{b}, \frac{\sum_{k=i}^{j} w^k}{s}, \frac{\delta^j}{b}\right) \leq P_{\text{giv}} \end{array} \right\}_{s \in \mathsf{S}}$

Since the platform is homogeneous, we denote by E_{stat} the static energy of all processors, and by $E_{dyn}(s)$ the dynamic energy consumed at speed s ($s \in \mathsf{S}$). Then, the recurrence is easy to justify: to compute $E(i,j,q)$, we create an interval from stages $\mathcal{S}^{\ell+1}$ to $\mathcal{S}^j$ that is assigned to one single processor, and we use the $q-1$ remaining processors to process stages $\mathcal{S}^i$ to $\mathcal{S}^\ell$. The initialization states that one single stage cannot be run on exactly more than one processor, and it returns the energy consumed by the processor in charge of interval $[i,j]$ so that the bound on the period is satisfied.

With many applications, for $a \in \{1,\ldots,A\}$ and $q \in \{0,\ldots,p\}$, let E_a^q the minimum energy consumed by q processors on the application a, computed by one the previous dynamic programming algorithms. If the period constraint cannot be fulfilled, or if the latency constraint cannot be fulfilled ($q > k_a^{\max}$), we set $E_a^q = +\infty$.

We recursively compute the value $E(a,q)$, which is the minimum energy consumed by exactly q processors on applications $1,\ldots,a$. The goal is thus to compute $\min_{1 \leq q \leq p} E(A,q)$. The recurrence relation can be expressed as:

$$E(a,q) = \min_{1 \leq r \leq q-1} \left(E(a-1, q-r) + E_a^r \right),$$

with the initialization:

$$E(1,q) = E_1^q, \forall 1 \leq q \leq p.$$

Indeed, when there is only one application left, the result is known from the previous dynamic programming algorithm. For several applications, we try to assign r processors to application a, and find the value of r which returns the lowest energy consumption. ■

7 Conclusion

In this report, we have studied the problem of mapping concurrent applications onto computational platforms according to three criteria: period, latency and energy. We restricted the study to the class of applications which have a pipeline structure, and we established the complexity of the problems for different variants of mapping strategies (one-to-one, interval and general mappings), and different types of platforms (ranking from fully homogeneous to fully heterogeneous).

First we focused on one-to-one and interval mappings with no resource sharing. We considered performance criteria, namely period or latency minimization. From this study of mono-criterion problems, one striking result is the impact of having multiple concurrent applications on the problem complexity.

Indeed, when several applications are in competition for resources, the period minimization problem turns out NP-hard for interval mappings with heterogeneous processors, homogeneous pipelines and without communication, while a polynomial algorithm had been found to solve the same problem with a single application. The same phenomenon happens for latency minimization with one-to-one mappings. For other period or latency minimization problems, either we were able to extend polynomial algorithms for the single application case, or the problem remained NP-complete. Considering bi-criteria problems, we were able to derive nice sophisticated multi-criteria polynomial algorithms, through the construction of bipartite graphs or the use of dynamic programming. Trade-offs were found to allow for an efficient albeit energy-aware execution. Finally, the most challenging tri-criteria problem period/latency/energy turned out to be NP-hard even with a single application on a fully homogeneous platform and no communication cost.

In order to handle processor sharing, we explained why it was mandatory to use a simpler model for the latency, and we discussed the use of the WAVEFRONT latency model. Thanks to a combination of two dynamic programming algorithms, we showed that finding an optimal interval mapping without reuse on fully homogeneous platforms can be done in polynomial time, while the same problem was shown to be NP-complete with the classical definition of latency. However, finding an optimal general mapping on any platform type, or finding any optimal interval mapping on speed-heterogeneous platforms, are NP-complete problems.

We believe that this exhaustive complexity analysis provides a solid theoretical foundation for the study of multi-criteria mappings of several concurrent applications, in particular when combining performance and energy optimization criteria.

On the practical side, we designed several heuristics in [5], as well as an integer linear program to compute the optimal solution (either interval-based or general) in possibly exponential time, for the WAVEFRONT latency model. The comparison of heuristics with and without processor sharing does confirm that sharing is most useful when: (i) the modes are not close to each other; and (ii) the static energy is high.

As future work, we envision to add replication to the mapping rules: a stage could be mapped onto several processors, each in charge of different data sets, in order to improve the period. This problem, partially investigated in [7], would become even more challenging in a framework accounting for energy issues. Also, it would be interesting to include the consumption induced by memory, disks, fans, and other devices, in the energy model. Finally, we would like to consider different application settings, as for instance applications that share some data paths. In this case, we expect the impact of resource sharing to be even more important, since mapping two such applications on the same resource may further reduce their period and latency.

Acknowledgment

A. Benoit and Y. Robert are with the Institut Universitaire de France. This work was supported in part by the ANR *StochaGrid* and *RESCUE* projects.

References

[1] K. Agrawal, A. Benoit, L. Magnan, and Y. Robert. Scheduling algorithms for workflow optimization. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE CS Press, May 2010. Also available as research report RR-LIP-2009-22 at http://graal.ens-lyon.fr/~abenoit/.

[2] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. M. Al-Hashimi. Overhead-conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, page 10518, Washington, DC, USA, 2004. IEEE Computer Society Press.

[3] M. A. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *Proceedings of SODA'98*, 1998.

[4] A. Benoit, P. Renaud-Goud, and Y. Robert. Performance and energy optimization of concurrent pipelined applications. Research Report RR-LIP-2009-27, LIP, ENS Lyon, France, September 2009. Available at http://graal.ens-lyon.fr/~abenoit. Short version appears in IPDPS'2010.

[5] A. Benoit, P. Renaud-Goud, and Y. Robert. Sharing resources for performance and energy optimization of concurrent streaming applications. Research Report 2010-05, LIP, ENS Lyon, France, February 2010. Available at http://graal.ens-lyon.fr/~abenoit/. Short version appears in SBAC-PAD'2010.

[6] A. Benoit and Y. Robert. Mapping pipeline skeletons onto heterogeneous platforms. *Journal of Parallel and Distributed Computing (JPDC)*, 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*, 2009. Available online at http://dx.doi.org/10.1007/s00453-008-9229-4.

[8] A. Benoit, Y. Robert, and E. Thierry. On the complexity of mapping linear chain applications onto heterogeneous platforms. *Parallel Processing Letters (PPL)*, 19(3):383–397, 2009.

[9] D. P. Bunde. Power-aware scheduling for makespan and flow. In *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 190–196, New York, NY, USA, 2006. ACM Press.

[10] J.-J. Chen and L. Thiele. Energy-efficient task partition for periodic real-time tasks on platforms with dual processing elements. In *Proceedings of International Conference on Parallel and Distributed Systems (ICPADS)*, pages 161–168, Washington, DC, USA, 2008. IEEE Computer Society Press.

[11] S. Cho and R. G. Melhem. On the interplay of parallelization, program performance, and energy consumption. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 21:342–353, 2010.

[12] M. Cole. Bringing Skeletons out of the Closet: A Pragmatic Manifesto for Skeletal Parallel Programming. *Parallel Computing*, 30(3):389–406, 2004.

[13] DataCutter. DataCutter Project: Middleware for Filtering Large Archival Scientific Datasets in a Grid Environment. http://www.cs.umd.edu/projects/hpsl/ResearchAreas/DataCutter.htm.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[15] F. Gruian and K. Kuchcinski. Lenes: task scheduling for low-energy systems using variable supply voltage processors. In *Proceedings of the Asia South Pacific Design Automation Conference (ASPDAC)*, pages 449–455, New York, NY, USA, 2001. ACM.

[16] S. L. Hary and F. Özgüner. Precedence-constrained task allocation onto point-to-point networks for pipelined execution. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 10(8):838–851, 1999.

[17] Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T. Boku, and D. Takahashi. Profile-based optimization of power performance by using dynamic voltage scaling on a pc cluster. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, page 340, Los Alamitos, CA, USA, 2006. IEEE Computer Society Press.

[18] T.-Y. Huang, Y.-C. Tsai, and E. T.-H. Chu. A near-optimal solution for the heterogeneous multi-processor single-level voltage setup problem. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, page 57, Los Alamitos, CA, USA, 2007. IEEE Computer Society Press.

[19] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, pages 197–202. ACM Press, 1998.

[20] N. T. Karonis, B. Toonen, and I. Foster. MPICH-G2: A grid-enabled implementation of the message passing interface. *Journal of Parallel and Distributed Computing (JPDC)*, 63(5):551–563, 2003.

[21] P. Langen and B. Juurlink. Leakage-aware multiprocessor scheduling. *Journal of Signal Processing Systems*, 57(1):73–88, 2009.

[22] M. P. Mills. The internet begins with coal. *Environment and Climate News*, page ., 1999.

[23] F. Rabhi and S. Gorlatch. *Patterns and Skeletons for Parallel and Distributed Computing*. Springer Verlag, 2002.

[24] J. Subhlok and G. Vondran. Optimal mapping of sequences of data parallel tasks. In *Principles and Practice of Parallel Programming (PPoPP)*, 1995.

[25] J. Subhlok and G. Vondran. Optimal latency-throughput tradeoffs for data parallel pipelines. In *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 1996.

[26] K. Taura and A. A. Chien. A heuristic algorithm for mapping communicating tasks on heterogeneous resources. In *Heterogeneous Computing Workshop*, pages 102–115. IEEE Computer Society Press, 2000.

[27] G. Varatkar and R. Marculescu. Communication-aware task scheduling and voltage selection for total systems energy minimization. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, page 510, Washington, DC, USA, 2003. IEEE Computer Society Press.

[28] N. Vydyanathan, U. Catalyurek, T. Kurc, P. Sadayappan, and J. Saltz. A duplication based algorithm for optimizing latency under throughput constraints for streaming workflows. In *Proceedings of International Conference on Parallel Processing (ICPP)*, pages 254–261, Washington, DC, USA, 2008. IEEE Computer Society.

[29] N. Vydyanathan, U. Catalyurek, T. Kurc, P. Saddayappan, and J. Saltz. Toward optimizing latency under throughput constraints for application workflows on clusters. In *Euro-Par'07*, LNCS 4641, pages 173–183. Springer Verlag, 2007.

[30] Q. Wu, J. Gao, M. Zhu, N. Rao, J. Huang, and S. Iyengar. On optimal resource utilization for distributed remote visualization. *IEEE Transactions on Computers (TC)*, 57(1):55–68, 2008.

[31] Q. Wu and Y. Gu. Supporting distributed application workflows in heterogeneous computing environments. In *Proceedings of International Conference on Parallel and Distributed Systems (ICPADS)*, pages 3–10, Washington, DC, USA, 2008. IEEE Computer Society Press.

[32] R. Xu, D. Mossé, and R. Melhem. Minimizing expected energy in real-time embedded systems. In *Proceedings of the ACM Int. Conf. on Embedded Software (EMSOFT)*, pages 251–254, 2005.

[33] R. Xu, D. Mossé, and R. Melhem. Minimizing expected energy consumption in real-time systems through dynamic voltage scaling. *ACM Trans. Comput. Syst.*, 25(4):9, 2007.

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

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

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