Performance and energy optimization of concurrent pipelined applications

Anne Benoit, Paul Renaud-Goud, Yves Robert

Performance and energy optimization of concurrent pipelined applications

Anne Benoit, Paul Renaud-Goud and Yves Robert

LIP, ENS Lyon, 46 Allée d'Italie, 69364 Lyon Cedex 07, France
UMR 5668 - CNRS - ENS Lyon - UCB Lyon - INRIA
{Anne.Benoit|Paul.Renaud-Goud|Yves.Robert}@ens-lyon.fr

September 2009

LIP Research Report RR-2009-27

Abstract

In this paper, 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 application with several stages, and processes successive data sets in pipeline mode, from the first to the last stage. We study the problem complexity on different target execution platforms, ranking from fully homogeneous platforms to fully heterogeneous ones. The goal is to select an execution speed for each processor, and then to assign stages to processors, with the aim of optimizing several concurrent optimization criteria. There is a clear trade-off to reach, since running faster and/or more processors leads to better performance, but the energy consumption is then very high. Energy savings can be done at the price of a lower performance, by reducing processor speeds or enrolling fewer resources.. We consider two mapping strategies: in one-to-one mappings, a processor is assigned a single stage, while in interval mappings, a processor may process an interval of consecutive stages of the same application. For both mapping strategies and all platform types, we establish the complexity of several multi-criteria optimization problems, whose objective functions combine period, latency and energy criteria. In particular, we exhibit cases where the problem is NP-hard with concurrent applications, while it can be solved in polynomial time for a single application. Also, we demonstrate the difficulty of performance/energy trade-offs by proving that the tri-criteria problem is NP-hard, even with a single application on a fully homogeneous platform.

Contents

1 Introduction

In this paper, we aim at optimizing the execution of several independent pipelined applications that execute concurrently on a given platform. Indeed, pipelined applications are becoming increasingly prevalent, see for instance [7, 18, 22]. 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.

We focus in this work on pipelined applications with the regular structure of a linear chain. Such applications are ubiquitous in streaming environments, as for instance video and audio encoding and decoding, DSP applications, image processing, and so on ([7, 18, 10, 21, 22]). Furthermore, the regularity of these applications render them amenable to a high-level parallel programming approach based on algorithmic skeletons ([6, 14]). Skeletons ease the task of the application developer and make it easy to tailor his/her specific problem to a target platform. In linear pipeline applications, a series of data sets enter the input stage and progress from stage to stage until the final result is computed. Each stage 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 first 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. Typical performance-related objectives for such pipelined operations are the *period* (which is defined as the inverse of the throughput) or the *latency* (also called response time) [16, 17, 19, 20, 3, 4, 21]. Formally, the period of a mapping is defined as the time interval required between the beginning of the execution 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. As for the latency, it is defined as 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. Period and latency already are conflicting objectives when executing a single application. 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.

In the recent years, another critical problem arose, namely the *energy consumption* of computational platforms. As an example, the Earth Simulator requires about 12 megawatts of peak power, and Petaflop systems may require 100 MW of power, nearly the output of a small power plant (300 MW). At \$100 per MegaWatt.Hour, peak operation of a petaflop machine may thus cost \$10,000 per hour [9]. Current estimates state that cooling costs \$1 to \$3 per watt of heat dissipated [15]. This is just one of the many economical reasons why energy-aware scheduling has proved to be an important issue in the past decade, even without considering battery-powered systems such as laptop and embedded systems.

The emphasis of this paper is on a multi-criteria approach, where efficient trade-offs must be found between performance-related objectives that are typical of pipelined applications, namely period and latency minimization, and the total energy consumed by enrolled resources. For this purpose, we consider multi-modal processors: each processor has a discrete number of speeds (or modes) of computation, which can be obtained by changing the processor frequency: the faster the speed, the less efficient energetically-speaking [11]. At the beginning of execution, we must decide at which speed each computer will operate, and this speed is then fixed for the whole execution. The energy-oriented objective is to minimize the total energy consumption of the platform; it is computed as the sum of the energy consumed by each processor, which is a

function of its speed (dynamic energy) and of a fixed overhead (static energy), similarly to the model in [12].

Our global aim 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 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 informally as follows: given a set of applications and a computational platform, which stage to assign to which processor? We consider two different mapping strategies: *one-to-one* mappings, for which each application stage is allocated to a distinct processor; and *interval* mappings, where each participating processor is assigned an interval of consecutive stages. These mapping strategies have been widely used in the literature when mapping one single application (see for instance [16, 17, 3]), and we extend them naturally to the mapping of several concurrent applications without allowing any processor sharing. This assumption is quite realistic from the point of view of the platform manager whose goal may be to secure an efficient (albeit concurrent) execution for each application, and is further motivated from a theoretical point of view in Section 3.3.

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, which constitute the most difficult problem instance.

Finally, we aim at optimizing several contradictory criteria, namely period, latency and energy, and we study all combination of these criteria. However, when taking energy into account, we always include the period in the combination, since the energy is an energy spent per time unit, which makes sense only in a pipelined execution of the application, while latency by its own takes only one single data set into account. Thus we consider two mono-criterion problems consisting in minimizing the period or the latency, and then two bi-criteria problems combining period/latency or period/energy, and finally the tri-criteria problem combining all three optimization criteria. Altogether, with two mapping strategies, three target platforms and five criteria combination, we have 30 problems to solve. A major contribution of this paper is to establish the complexity of all these problems in the context of multiple concurrent pipelined applications.

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 [16] and [17]) and later for heterogeneous platforms (see [3, 4]). 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.

The paper is organized as follows. We start by illustrating and motivating the problem with a simple example in Section 2. Then we describe the framework in Section 3. The next two sections constitute the heart of the paper: we assess the complexity of all problem instances. Results for period or latency minimization are reported in Section 4, while results for multi-criteria problems are presented in Section 5. Finally we conclude in Section 6.

2 Motivating example

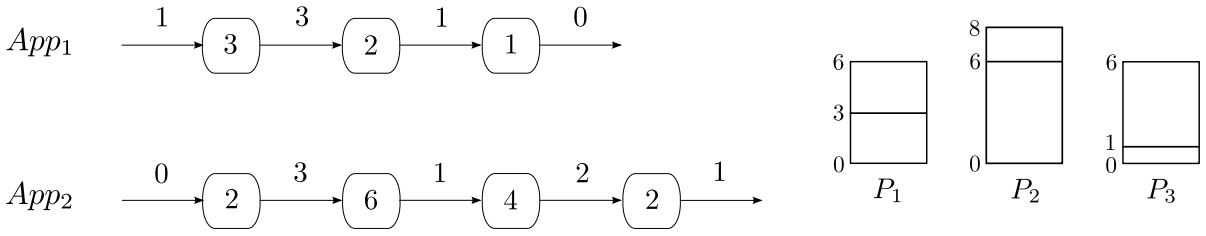

Figure 1: Example with two applications and three multi-modal processors

In this small example we have two applications and three processors, as shown on Figure 1. We restrict to interval mappings, where a processor can be assigned only a set of consecutive stages of a single application. The first stage of App_1 receives a data of size 1, then computes 3 operations, and finally sends a data of size 3 to the second stage, and so on. If both stages are assigned to the same processor, there will be no communication cost to pay; otherwise this cost will depend on the communication volume (3 in this case) and on the link bandwidth between the corresponding processor pair. 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. The energy consumption of a processor is equal to the square of its speed, which is quite a realistic assumption (see Section 3.5 for more details on the model for energy consumption). Finally, all communication link bandwidths are set to 1.

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. 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. 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:

$$\max\left(\max\left(\frac{1}{1}, \frac{3+2+1}{6}, \frac{0}{1}\right), \max\left(\max\left(\frac{0}{1}, \frac{2+6}{8}, \frac{1}{1}\right), \max\left(\frac{1}{1}, \frac{4+2}{6}, \frac{1}{1}\right)\right)\right) = 1 \qquad (1)$$

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. 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 example (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:

$$\max\left(\frac{1}{1}+\frac{3+2+1}{6}+\frac{0}{1}, \frac{0}{1}+\frac{2+6+4+2}{8}+\frac{1}{1}\right) = 2.75 \quad (2)$$

In Equation (2) we simply compute the longest execution path for each application. 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 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:

$$\max\left(\max\left(\frac{1}{1}, \frac{3+2+1}{3}, \frac{0}{1}\right), \max\left(\frac{0}{1}, \frac{2+6+4+2}{1}, \frac{1}{1}\right)\right) = 14$$

As expected, running at a slower pace 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 $6^2 + 8^2 + 6^2 = 136$.

3 Framework

We start with a formal description of the applicative framework (Section 3.1) and the target execution platform (Section 3.2). Next in Section 3.3, we introduce and motivate the mapping strategies. We are then ready to formally describe the performance objective criteria (period and latency) in Section 3.4, and then to finally discuss the energy model in Section 3.5.

3.1 Applicative framework

As shown in Figure 2, 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 $1 \leq k \leq n_a$, δ_a^k is the size of the output data of $\mathcal{S}_a^k$, the k^{th} stage of application a and w_a^k is its computation requirement. The first stage $\mathcal{S}_a^1$ of each application, $1 \leq a \leq A$, receives an input of size δ_a^0 from the outside world, while the last stage of each application $\mathcal{S}_a^{n_a}$ returns the result (of size $\delta_a^{n_a}$) to the outside world.

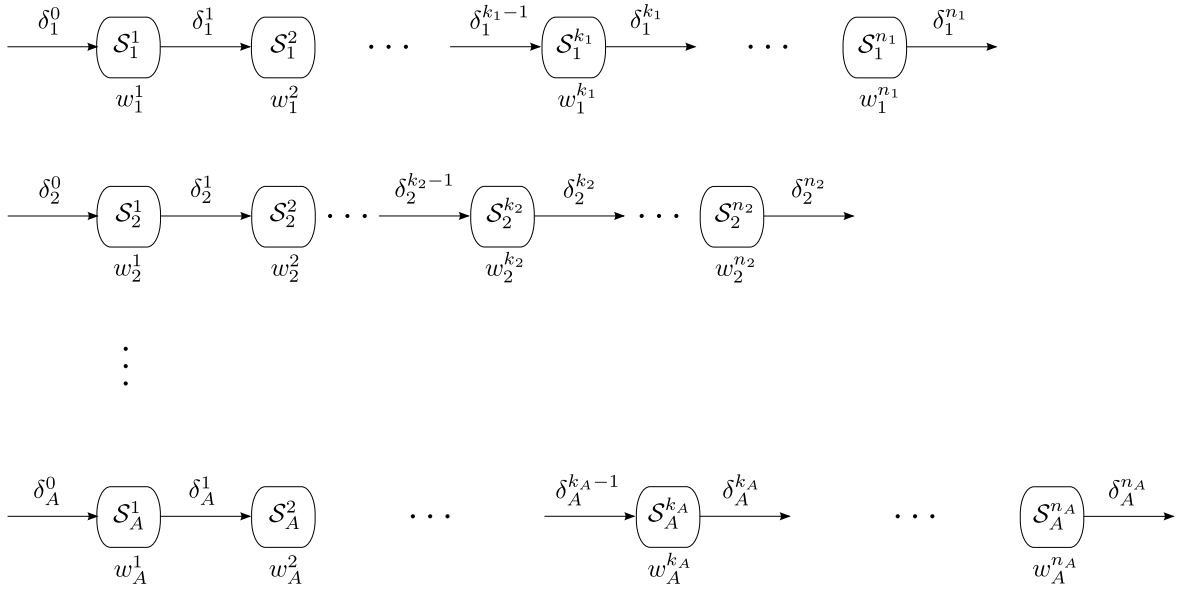

Figure 2: Notations

3.2 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 $b_{u,v}$. For simplification, we assume that $2A$ additional processors $P_{\mathsf{in}_1}, \ldots, P_{\mathsf{in}_A}$ and $P_{\mathsf{out}_1}, \ldots, P_{\mathsf{out}_A}$ are devoted to input/output operations of the applications (in fact these additional processors are virtual processes that may well be shared by the same physical resource). Initially, for each $a \in \{1, \ldots, A\}$, the input data for each task of the application a resides on P_{in_a}, while all results must be returned to and stored on P_{out_a}. These special processors are all connected to the p processors of the target platform.

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 3.3 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 2) or serialized. With parallel operations, we have the *overlap* model that corresponds to multi-threaded communication libraries such as MPICH2 [13]. With sequential operations, we have the *no-overlap* model that is well-suited to single-threaded programs. Afterwards, we prove the theorems for the overlap model, and when nothing more is said about communication model in the theorem proofs, they are valid for the both communication models.

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 $s_u \in \mathsf{S_u}$ 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* 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* 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.

3.3 Mapping strategies and scheduling

We consider two mapping strategies, *one-to-one* and by *interval*. *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 [16, 17, 3, 21, 22] among others.

We point out that both one-to-one and interval mappings forbid any processor sharing, or re-use, across applications. We could introduce *general* mappings that would allow any processor to execute any number of stages, consecutive or not, taken from one or several applications. However, there are several reasons, both practical and theoretical, to restrict to interval mappings:

- On the practical side, 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.
- On the theoretical side, there are two problems with general mappings:
  1. they immediately lead to NP-hard optimization problems, even for the simplest mono-criterion problem: period minimization for a single application mapped onto homogeneous and uni-modal processors, paying no communication cost (straightforward reduction from 2-PARTITION);
  2. they lead to intricate scheduling problems for period/latency bi-criteria problems.

The latter problem is the most important, although we discovered it only quite recently [1]. Basically, even when the mapping is given, scheduling the execution is a problem of combinatorial nature. 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. On the contrary, with interval mappings, we have two key properties: (i) the execution graph is acyclic, meaning that data leaving one processor never returns to that processor; and (ii) each processor has at most one incoming and one outgoing communication. Once the mapping has been determined, these two properties allow for a straightforward scheduling: each operation is executed as soon as possible.

3.4 Performance optimization criteria

We are now ready to formally define the *period* and the *latency* of one-to-one and interval mappings. Because there is no processor sharing, we can focus on a single application.

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:

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

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:

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

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

$$L = \frac{\delta^0}{b_{\mathsf{al}(0),\mathsf{al}(1)}} + \sum_{j=1}^{m} \left(\sum_{i=d_j}^{e_j} \frac{w^i}{s_{\mathsf{al}(d_j)}} + \frac{\delta^{e_j}}{b_{\mathsf{al}(d_j),\mathsf{al}(e_j+1)}} \right) \tag{5}$$

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 $a \in \{1, \ldots, A\}$. 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

$$X = \max_{a \in \{1,\ldots,A\}} W_a \cdot X_a \tag{6}$$

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 $W_a \cdot X_a$ represents the slowdown factor of application a, and X corresponds to the maximum stretch [2].

3.5 Energy model

The last criterion is the energy consumption of the platform, which is defined as the sum of the energy $E(u)$ consumed by each processor P_u enrolled in the mapping. We assume that $E(u)$ consists of a static part and of a dynamic part: $E(u) = E_{stat}(u) + E_{dyn}(s_u)$. 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 at which the processor is running. On the contrary, the dynamic part $E_{dyn}(s_u)$ is of the form $E_{dyn}(s) = s^\alpha$, where $\alpha > 1$ is an arbitrary rational number. It is sometimes assumed that $\alpha = 2$ [12], as we did in the example of Section 2, but all our results hold for any value of α.

The energy $E(u)$ is an energy consumed per time unit, so it must be associated with a duration. However, the execution of a pipelined application with arbitrarily many consecutive 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.

4 Complexity of mono-criterion problems

Note that since we do not consider energy minimization issues in our mono-criterion optimization problems, we can systematically run processors at their highest speed, and thus use classical results established in a context with no energy.

4.1 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 with equivalent pipelines and without communication.

4.1.1 One-to-one mappings

Communication homogeneous platforms

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 [3], 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:

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

because it is equal to the product of W_a by the cycle-time of some processor P_u executing some stage S_a^k . First we compute the set $\mathcal{T}$ and sort its elements into an array $\mathcal{T}_A$. Then we binary search the array $\mathcal{T}_A$ for 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 described below. 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 than T, and 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 $|\mathcal{T}| \leq n_{max}Ap$, where $n_{max} = \max_{a \in \{1,\ldots,A\}} n_a$, hence the total computation time is $O((n_{max}Ap + cost_{GA}) \log(n_{max}Ap))$, 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 application, 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 A, and assume that it has assigned stage $\mathcal{S}_{a_1}^{k_1}$ to P_1. Note first that the

Algorithm 1: Greedy-Assignment(T)

begin
  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 $\mathcal{S}_a^k$ s.t. $W_a \cdot \max(\frac{\delta_a^{k-1}}{b_a}, \frac{w_a^k}{s_u}, \frac{\delta_a^k}{b_a}) \leq T$
    Assign $\mathcal{S}_a^k$ to P_u
    Mark $\mathcal{S}_a^k$ as already assigned
    if *no stage found* **then**
      return *"failure"*
    end
  end
  return *"success"*
end

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 A. Then we exchange the assignments of P_1 and P_u in A. As P_u is faster than P_1, which could process $\mathcal{S}_{a_1}^{k_1}$ in time in the assignment 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 A 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 $N \leq p$ and $N \leq n_{max}A$, the complexity of the whole algorithm is $O((n_{max}Ap)^2 \log(n_{max}Ap))$, which is indeed polynomial in the problem size.

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

Fully heterogeneous platforms

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

Proof As the problem was already NP-complete with one single application [3], it remains NP-complete with concurrent applications.

4.1.2 Interval mapping

Fully homogeneous platforms

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

Proof A polynomial algorithm has already been found to exhibit the minimal period with one application in this case under a no-overlap communication model (see [4]), that can be easily extend to our overlap model, so the following proof is valid for both models.

We exhibit an algorithm which finds such a mapping for concurrent applications thanks to the previous polynomial algorithm, and we show then its validity.

Algorithm 2:

Assign all stages of each application to 1 processor
Compute the period of all applications
for $a \leftarrow (p - A)$ **to** p **do**
  Find an application a' such that $W_{a'} \cdot T_{a'}$ is maximum
  Add 1 processor to this application
  Compute the new period $T_{a'}$ of this application
end

First here are some notations:

- $(k^u_{a,i})$ is a A-tuple which represents the processor distribution among the applications at step i.
- $(k^o_{a,i})$ 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.
- $T(d) = \max_{a \in \{1,\dots,A\}} W_a \cdot T_a(d_a)$ where d is a A-uplet.

Now the proof:

- $(k^u_{a,A})$ is the best distribution with A processors, because it is the only one.
- Let assume that $(k^u_{a,i})$ is optimal with i processors. We want $(k^u_{a,i+1})$ to be an optimal distribution with $i + 1$ processors.
  - Either:
    $$\exists a, k^o_{a,i+1} < k^u_{a,i}$$
    By construction:
    $$\exists i' < i, T((k^u_{a,i'})) = W_a \cdot T_a(k^u_{a,i'}) = W_a \cdot T_a(k^o_{a,i+1})$$
    Now, because every T_a and $x \mapsto W_a \cdot x$ are non-decreasing,
    $$T((k^u_{a,i+1})) \leq T((k^u_{a,i})) \leq T((k^u_{a,i'}))$$
    and by definition,
    $$W_a \cdot T_a(k^o_{a,i+1}) \leq T((k^o_{a,i+1}))$$
    So
    $$T((k^u_{a,i+1})) \leq T((k^o_{a,i+1}))$$
  - 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:
      $$\exists a' \neq a, k^u_{a',i+1} = k^u_{a',i} + 1$$

By construction:

$$\begin{aligned} T((k^u_{a,i})) &= f_{a'}(T_{a'}(k^u_{a',i})) \\ &= f_{a'}(T_{a'}(k^o_{a',i+1})) \quad \text{because} \quad k^u_{a',i} = k^o_{a',i+1} \\ &\leq T((k^o_{a,i+1})) \end{aligned}$$

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 recurrence, the algorithm finds an optimal solution to map p processors into A applications.

The complexity of computing the period of an application with n processors, keeping the intermediate results with $n-1$ processors, is bounded by $O(n^3 p)$ [3]. That is why the complexity of the Algorithm 2 is bounded by $O(n^3 p^2)$.

Communication homogeneous platforms

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 [3], it remains NP-complete with concurrent applications.

Heterogeneous processors and homogeneous pipelines without communication

This case is more interesting, because a polynomial algorithm exists to find an interval mapping which minimizes the period of one single application [4]; however, the problem becomes NP-complete with several concurrent applications.

Theorem 5 *With more than one application, heterogeneous processors, homogeneous pipelines without communication, the optimal interval mapping which minimizes* $\max_{a\in\{1,\ldots,A\}} T_a$ *is a NP-complete problem (in the strong sense).*

Proof 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 [8]. 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 $\sum_{i\in I_j} a_i = B$?

  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: let m identical pipelines such that each pipeline 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 global 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, for each $j \in \{1, \ldots, m\}$, $I_j = \{a'_{1,j}, a'_{2,j}, a'_{3,j}\}$. For each $j \in \{1, \ldots, m\}$, we assign the $a'_{1,j}$ first consecutive stages of the application j to the (a) processor whose speed is equal to $a'_{1,j}$, 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 global 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 processors speed, 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 on 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 3 processors mapped into one application. So $\mathcal{I}_1$ has clearly a solution.

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

Theorem 6 *With more than one application, heterogeneous processors, homogeneous pipelines without communication, the problem finding the optimal interval mapping which minimizes* $\max_{a \in \{1,\ldots,A\}} W_a \cdot T_a$ *is NP-complete (in the strong sense).*

Proof 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 theorem.

Theorem 7 *With more than one application, heterogeneous processors, homogeneous pipelines without communication, the problem of finding the optimal interval mapping which minimizes* $\max_{a \in \{1,\ldots,A\}} T_a/T_a^*$ *is NP-complete (in the strong sense).*

Proof 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 $\max_{a \in \{1,\ldots,A\}} T_a$.

4.2 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.

4.2.1 One-to-one mapping

Fully homogeneous platforms

Theorem 8 *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.

Heterogeneous processors, no communication, homogeneous pipelines This is an interesting case, because a polynomial algorithm exists to find a one-to-one mapping which minimizes the latency of one single application [5]; however, the problem becomes NP-complete with several concurrent applications.

Theorem 9 *With more than one application, heterogeneous processors, homogeneous pipelines without communication, the problem of finding the optimal one-to-one mapping which minimizes* $\max_{a\in\{1,\ldots,A\}} L_a$ *is NP-complete (in the strong sense).*

Proof 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: let m identical pipelines such that each pipeline is 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 (a) 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 $\sum_{i\in I_j} a_i \leq B$. As $\sum_{i=1}^{m} a_i = mB$, 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.

Theorem 10 *With more than one application, heterogeneous processors, homogeneous pipelines without communication, the problem of finding the optimal one-to-one mapping which minimizes* $\max_{a\in\{1,\ldots,A\}} W_a \cdot L_a$ *is NP-complete (in the strong sense).*

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

Theorem 11 *With more than one application, heterogeneous processors, homogeneous pipelines without communication, the problem of finding the optimal one-to-one mapping which minimizes* $\max_{a\in\{1,\ldots,A\}} L_a/L_a^*$ *is NP-complete (in the strong sense).*

Proof The proof is the same as the first one, but we ask now whether it is possible to realize a global latency of KB, 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 $\max_{a\in\{1,\ldots,A\}} L_a$.

4.2.2 Interval mapping

Communication homogeneous platforms

Theorem 12 *On communication homogeneous platforms the optimal interval mapping which minimizes the latency can be determined in polynomial time.*

Proof

- With a single application the optimal mapping is to map the whole application onto one processor: if two processors compute two parts of the application, mapping the entire application onto the fastest processor will reduce the computation time and the communications cost.
- So 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:

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

  As $|\mathcal{L}| = Ap$, the complexity of our algorithm is $O((Ap+A^2)\log(Ap))$, that we can simplify in $O(Ap\log(Ap))$.

Fully heterogeneous platforms

Theorem 13 *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 [5], it remains NP-complete with several concurrent applications.

4.3 Summary

Table 1 summarizes all mono-criterion complexity results. Each column corresponds to a platform type: **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 homogeneous pipelines without communication.

The two special entries denoted with (*) are problem instances which could be solved in polynomial time for a single application, but becomes NP-hard when several ones. Remaining entries correspond to polynomial algorithms that were already existing for a single application and that have been extended for several one. Finally, all results apply to both the overlap and no-overlap models, and to all objective functions introduced in Section 3.4: more precisely, polynomial problems remain polynomial for arbitrary weights W_a in Equation (6), while NP-complete problems are already difficult with $W_a = 1$.

	proc-hom	**proc-het**		
	com-hom	**special-app**	**com-hom**	**com-het**
Period - *one-to-one*	polynomial (binary search)			NP-complete
Period - *interval*	polynomial (dyn. prog. + greedy)	NP-complete(*)	NP-complete	
Latency - *one-to-one*	polynomial	NP-complete(*)		NP-complete
Latency - *interval*	polynomial (binary search)			NP-complete

Table 1: Complexity results for mono-criterion optimization problems

5 Complexity of multi-criteria problems

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.

5.1 Period/latency minimization

In this section again, we are not concerned with energy minimization issues, so, similarly to results of Section 4, 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.

We exhibit a dynamic programming algorithm to find an optimal interval mapping on fully homogeneous platforms, which is the only case where one of the mono-criterion problem is polynomial.

Fully homogeneous platforms

One-to-one mapping

Theorem 14 *On fully homogeneous platforms the problem of finding the one-to-one mapping, which minimizes the latency for a given period, or the opposite, is polynomial.*

Proof On such platforms all mappings are equivalent, so the problem is polynomial.

Interval mapping

Theorem 15 *With one application, on fully homogeneous platforms the optimal 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:

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

This relation is effective 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 $i \in \{1, \ldots, n\}$:

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

- Too much processors; if $q > 1$:

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

Finally we aim at computing:

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

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

We use a binary search to find a mapping, which minimizes the period for a given latency.

On the one side, the minimum period belongs to the set:

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

On the other side, 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 $T^2_{lim} > T_{lim}$. 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.

As $|\mathcal{T}| = \frac{n(n+1)}{2} + n$, 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:

$$\mathcal{T} = \left\{ \frac{\delta^{i-1}}{b} + \frac{\sum_{k=i}^{j} w^k}{s} + \frac{\delta^j}{b}, i \in \{1, \dots, n\}, j \in \{i, \dots, n\} \right\}$$

.

Theorem 16 *With several applications, on fully homogeneous platforms the optimal mapping which minimizes the latency $L = \max_{a \in \{1,\dots,A\}} W_a \cdot L_a$ for a bounded period by application, or the period $T = \max_{a \in \{1,\dots,A\}} W_a \cdot T_a$ for a bounded latency by application can be determined in polynomial time.*

Proof There we can reuse the Algorithm 2.

If we want to solve the first problem, we run the version of the previous algorithm which computes the minimum latency for the bounded period $T_{a'}$ to compute the latency of the application a', and if we want to solve the second problem, we run the version of the previous algorithm which computes the minimum period for the bounded latency $L_{a'}$ to compute the period of the application a'.

That leads us to a complexity in $O((np)^2 \log(n))$ for the period minimization with a bounded latency, and $O((np)^2)$ for the latency minimization with a bounded period.

Heterogeneous processors, homogeneous pipeline, no communication

Theorem 17 *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.

5.2 Period/energy minimization

Contrary to the previous problems, we can no more operate the proofs with uni-modal processors because of the energy.

We show that finding an optimal one-to-one mapping on communication homogeneous platforms can be done by finding a minimum weight matching in a graph, which proves that our problem is polynomial. In addition after exhibiting a dynamic programming algorithm that finds an optimal interval mapping on fully homogeneous platforms for a single application, we extend it to many applications.

5.2.1 Preliminary results with one application: interval mapping

Theorem 18 *On fully homogeneous platforms the optimal interval mapping, which minimizes the consumed energy for a given period, can be determined in polynomial time.*

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

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

with the initialization:

- $E(i,i,q) = +\infty \;\text{ if }\; q > 1$
- Defining $\mathcal{F}_i^j = \left\{E_{dyn}(s_\ell) + E_{stat}, \max\left(\frac{\delta^{i-1}}{b}, \frac{\sum_{k=i}^{j} w^k}{s_\ell}, \frac{\delta^j}{b}\right) \leq T, \ell \in \{1,\dots,m\}\right\}$,

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

The complexity of this dynamic programming algorithm is bounded by $O(n^2(p+m))$: $O(n^2m)$ to compute the $E(i,j,1)$, $i \in \{1,\dots,n\}$,$j \in \{i,\dots,n\}$ and $O(n^2p)$ to compute the $E(1,i,k)$, $i \in \{1,\dots,n\}$ and $k \in \{2,\dots,p\}$.

To prove the theorem in the no-overlap model, we must 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 $\mathcal{F}_i^j$.

5.2.2 Results with many applications

One-to-one mapping

Theorem 19 *On communication homogeneous platforms, a one-to-one mapping, which minimizes the power consumption for a given period by application, can be determined in polynomial time.*

Proof We build a bipartite graph, and find a minimum weighted bipartite matching in it, that means a matching in which the sum of the edges weights are minimum.

Let $G = (U, V, E)$ a bipartite graph, where 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 ∞ if the processor cannot execute the stage within the period, and else the energy consumed by the processor when it is running in the smallest mode which allows it to run the stage within the period.

Finally finding a minimum weighted bipartite matching will give us, in polynomial time, the minimum power consumption needed to execute the all stages in a one-to-one mapping.

The complexity of the Hopcroft-Karp algorithm, which solves this problem, is in $O(\sqrt{UV}E)$, so in $O\left((np)^{\frac{3}{2}}\right)$ in our case.

Theorem 20 *On fully heterogeneous platforms, finding a one-to-one mapping, which minimizes the power consumption for given periods or the global period for a given energy, is a NP-complete problem.*

Proof This comes directly from the NP-completeness of the period minimization problem.

Interval mapping

Theorem 21 *On fully homogeneous platforms, an interval mapping, which minimizes the power consumption for a given period by application, can be determined in polynomial time.*

Proof For $a \in \{1, \ldots, A\}$ and $k \in \{0, \ldots, p\}$, let E_a^k the minimum energy consumed by k processors on the application a, computed by the previous dynamic programming algorithm. If it fails, $E_a^k = +\infty$. We note $E(a, k)$ the minimum energy consumed by k processors on the applications $1, \ldots, a$, so we are looking for $\min_{k \in \{1,\ldots,p\}} E(A, k)$. This energy can be computed recursively, thanks to the relations:

$$\forall k \in \{1, \ldots, p\} \begin{cases} \forall a \in \{2, \ldots, A\} & E(a, k) = \min_{q \in \{0,\ldots,k-1\}} \left(E_a^q + E(a-1, k-q)\right) \\ E(1, k) = E_1^k \end{cases}$$

If we first compute the E_a^q for each $a \in \{1, \ldots, A\}$ and $q \in \{1, \ldots, p\}$, then call the previous algorithm, the whole complexity is in $O(An^3p^2 + Ap)$, so $O(An^3p^2)$.

Theorem 22 *On not fully homogeneous platform, finding an interval mapping, which minimizes the power consumption for a given period by application or the global period for a given energy, is a NP-complete problem.*

Proof Under those hypothesis the period minimization problem is NP-complete. So the bi-criteria minimization problem is.

5.3 Period/latency/energy minimization

We can straight away restrain our study field: as the bi-criteria period/latency minimization problem is NP-complete on any other platform type than fully homogeneous, this tri-criteria is already NP-complete for those platform types.

Moreover we study both uni-modal and multi-modal processors, since their difference becomes more remarkable on the complexity results.

5.3.1 Uni-modal processors

Theorem 23 *On fully homogeneous platforms, finding a one-to-one mapping that solves the tri-criteria problem can be done in polynomial time.*

Proof On those platforms all mappings are equivalent, and we do not have to make a choice on the processors speed.

Theorem 24 *On fully homogeneous platforms, finding an interval mapping that solves the tri-criteria problem can be done in polynomial time.*

Proof For one application, we wrote a polynomial time algorithm that minimizes the latency for a bounded period and a given maximum number of processors, or minimizes the period, for a bounded latency and a given maximum number of processors.

Let us come back to several concurrent applications and try to solve the problem: minimization of $T = \max_{a \in \{1,\dots,A\}} W_a \cdot T_a$ for a given array of latency and a given consumed energy. The given energy gives us the maximum number of available processors, as they have only one mode. We can then once more reuse the Algorithm 2. Instead of computing the new period $T_{a'}$ of the application a', we run the algorithm that minimizes the period for a given latency, which is here the a'^{th} element of the given latency array.

For an application a', the complexity of computing the minimum latency for a given period and q processors, knowing the minimum latency for the same given period and $q-1$ processors is in $O(n^2)$. The complexity of the whole algorithm is thus $O(n^2 p)$.

We consider now the variant of the tri-criteria problem: latency minimization for given periods by application and a given consumed energy. It can be solved in the same way, with the Algorithm 2. We firstly build the sorting period set once, then for one application, the complexity of computing the minimum period for a given latency and a given number of processors is $O(n^2 \log n)$ if we know the result with less processors. The final complexity is $O(n^2 p \log n)$.

The last variant (energy minimization for given periods and latency by application) is simpler: we iterate on the applications. While there are remaining applications and processors, we compute the minimum number of needed processors to obtain a latency of the current application lesser than the latency given in the latency array, and a period lesser than the one given in the period array (with one of the bi-criteria problem; we choose rather the latency minimization for a given period, because of its better complexity). The sum of all energies is the energy we are looking for.

The complexity of this algorithm is $O(n^2 p)$.

Theorem 25 *On not fully homogeneous platforms, finding an interval mapping or a one-to-one mapping that solves the tri-criteria problem is a NP-complete problem.*

Proof The period/latency minimization problem is already NP-complete under those hypothesis.

5.3.2 Multi-modal processors

Theorem 26 *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 [8]. 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 $S = \sum_{i=1}^{n} a_i$.

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

  $$\forall i \in \{1, \ldots, n\} \quad \left\{ \begin{array}{l} s_{2i-1} = K^i \\ s_{2i} = K^i + \frac{a_i \cdot X}{K^{i\alpha}} \end{array} \right.$$

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

  Remember that α is the exponent used in the computation of the energy (see Section 3.5). 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 choose K such that for each $j \in \{2, \ldots, n\}$,

  $$K^{j\alpha} > \sum_{i=1}^{j-1} K^{i\alpha} + \alpha \cdot \left(\frac{S}{2} - \frac{1}{2} \right) \quad \text{and} \quad K^{j\alpha+1} > \sum_{i=1}^{j} K^{i\alpha} + \left(\frac{K^{\alpha+1}}{K^{j-1}} \cdot a_{j-1} + 1 - \frac{S}{2} \right)$$

  We can choose such a K, because the degrees of K in the left hands are strictly higher than those of the right hands. For each $j \in \{2, \ldots, n\}$ and each $0 < X < 1$:

  $$K^{j\alpha} > \sum_{i=1}^{j-1} K^{i\alpha} + \alpha X \cdot \left(\frac{S}{2} - \frac{1}{2} \right) \quad \text{and} \quad K^{j\alpha+1} > \sum_{i=1}^{j} K^{i\alpha} + X \cdot \left(\frac{K^{\alpha+1}}{K^{j-1}} \cdot a_{j-1} + 1 - \frac{S}{2} \right)$$

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

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

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

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

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

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

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

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

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

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

and for all $i \in \{2, \ldots, n\}$, $|f^{L_i}(X)| < X \cdot \frac{1}{2}$. Finally, we have to decide for the latency, the energy and the 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 $i \in \{1, \ldots, n\}$:

$$E^* = \sum_{i=1}^{n} s_{2i-1}^{\alpha} = \sum_{i=1}^{n} K^{i\alpha}$$

$$L^* = \sum_{i=1}^{n} \frac{w_i}{s_{2i-1}} = E^*$$

We ask whether it is possible to achieve an energy $E^o = E^* + X \cdot (S/2 + 1/2)$, a latency $L^o = L^* - (S/2 - 1/2)$ 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 $i \in \{1, \ldots, n\} \setminus I$, stage $\mathcal{S}_i$ is executed at speed s_{2i-1}. The mapping consumes an energy E and has a latency L, where:

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

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 $i \in \{1, \ldots, n\}$, stage $\mathcal{S}_i$ is executed at speed either s_{2i-1} or s_{2i}. Let $(\mathcal{P}_j)$ be the property: for each $i \in \{j, \ldots, n\}$, 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:

$$E \geq 2 \cdot s_{2n-1}^{\alpha} > K^{n\alpha} + \sum_{i=1}^{n-1} K^{i\alpha} + \alpha X \cdot \left(\frac{S}{2} + \frac{1}{2}\right)$$
$$> E^o$$

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

$$L \geq \frac{w_n}{s_{2n-2}} \geq K^{n\alpha+1} - \frac{K^{\alpha+1}}{K^{n-1}} \cdot a_{n-1}X + f^{L_i}(X)$$
$$> \sum_{i=1}^{n} K^{i\alpha} + X \cdot \left(\frac{K^{\alpha+1}}{K^{n-1}} \cdot a_{n-1} + 1 - \frac{S}{2}\right) - \frac{K^{\alpha+1}}{K^{n-1}} \cdot a_{n-1}X + f^{L_i}(X)$$
$$> \sum_{i=1}^{n} K^{i\alpha} - X \cdot \left(\frac{S}{2} - \frac{1}{2}\right) + \left(\frac{X}{2} + f^{L_i}(X)\right) > L^o$$

We conclude that $(\mathcal{P}_n)$ is true. We now proceed by induction. If for some $j \in \{3, \ldots, n\}$, $(\mathcal{P}_j)$ 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 $\{1, \ldots, n\}$ such that the processor that is assigned the stage $\mathcal{S}_i$ is running at speed s_{2i}. Then for each $i \in \{1, \ldots, n\} \setminus I$, the processor that is assigned stage $\mathcal{S}_i$ is running at speed s_{2i-1}. The consumed energy is $E = E^* + \sum_{i \in I} \left(\alpha a_i X + f_i^E(X)\right)$. But $E \leq E^o$, 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 $\sum_{i \in I} a_i < \frac{S}{2} + \left(\frac{1}{2} + \frac{1}{2}\right)$. As the a_i are integers, we derive 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.

Theorem 27 *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 26. 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:

$$\begin{cases} \forall i \in \{1, \ldots, n\} & w_{2i-1} = K^{i(\alpha+1)} \\ \forall i \in \{1, \ldots, n-1\} & w_{2i} = K^{(n+1)(\alpha+1)} \end{cases}$$

We use $2n-1$ identical processors, that can run $2n+1$ modes, such that:

$$\begin{cases} \forall i \in \{1,\dots,n\} & \begin{cases} s_{2i-1} = K^i \\ s_{2i} = K^i + \frac{a_i \cdot X}{K^{i\alpha}} \end{cases} \\ s_{2n+1} = K^{n+1} \end{cases}$$

We search for an interval mapping, whose energy does not exceed $E^o = (n-1)K^{(n+1)\alpha} + E^* + X \cdot (S/2+1/2)$, whose latency does not exceed $L^o = (n-1)K^{(n+1)\alpha} + L^* - (S/2-1/2)$, 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 other stages. All we have to do next is to find a one-to-one mapping of the unassigned stages. while 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 26.

5.4 Summary

All complexity results are summarized in Table 2. Just as before, all results apply to both the overlap and no-overlap models, and to all objective functions introduced in Section 3.3.

For each bi-criteria problem it is polynomial when the two mono-criterion problems are polynomial and turns NP-complete when at least one of the mono-criterion problem is NP-complete.

The tri-criteria problem is NP-complete with multi-modal processors even on fully homogeneous platforms.

	proc-hom	**proc-het**		
	com-hom	**special-app**	**com-hom**	**com-het**
Period/Latency - *both*	polynomial	NP-complete		
Period/Energy - *one-to-one*	polynomial (minimum matching)			NP-complete
Period/Energy - *interval*	polynomial (dyn. prog.)	NP-complete		
Period/Latency/Energy - *both*	NP-complete			

Table 2: Complexity results for multi-criteria optimization problems with multi-modal processors

6 Conclusion

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

First 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 put a particular emphasis on problems involving both performance and energy criteria. 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.

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.

As future work, on the theoretical side, we envision to add replication into the mappings: a stage could be mapped onto several processors, each in charge of different data sets, in order to improve the period, as was investigated in [4]. The problem would become even more challenging in a framework accounting for energy issues. On a more practical side, we plan to design some polynomial-time heuristics to solve the tri-criteria optimization problem in a general framework, in order to offer practical solutions to a difficult problem. It would also be challenging, both from a theoretical and a practical perspective, to assess the impact of processor sharing between applications, for situations in which it would be allowed by the platform manager.

References

[1] K. Agrawal, A. Benoit, L. Magnan, and Y. Robert. Scheduling algorithms for workflow optimization. Research Report 2009-22, LIP, ENS Lyon, France, July 2009. Available at `http://graal.ens-lyon.fr/~yrobert/`. Short version submitted to IPDPS'2010.

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

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

[4] 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.

[5] 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.

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

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

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[9] R. Ge, X. Feng, and K. W. Cameron. Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters. In *ACM/IEEE conference on Supercomputing (SC'05)*, page 34. IEEE Computer Society, 2005.

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

[11] 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 *International Parallel and Distributed Processing Symposium IPDPS'2006*. IEEE Computer Society Press, 2006.

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

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

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

[15] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan. Temperature-aware microarchitecture: modeling and implementation. *ACM Transactions on Architecture and Code Optimization*, 1(1):94–125, 2004.

[16] J. Subhlok and G. Vondran. Optimal mapping of sequences of data parallel tasks. In *Proc. 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 1995.

[17] J. Subhlok and G. Vondran. Optimal latency-throughput tradeoffs for data parallel pipelines. In *ACM Symposium on Parallel Algorithms and Architectures*, 1996.

[18] 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.

[19] 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.

[20] N. Vydyanathan, U. Catalyurek, T. Kurc, P. Saddayappan, and J. Saltz. A duplication based algorithm for optimizing latency under throughput constraints for streaming workflows. In *ICPP'2008, the Int. Conf. on Parallel Processing*, pages 254–261. IEEE Computer Society Press, 2008.

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

[22] Q. Wu and Y. Gu. Supporting distributed application workflows in heterogeneous computing environments. In *14th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE Computer Society Press, 2008.