Scheduling algorithms for workflow optimization

Kunal Agrawal[1], Anne Benoit[2], Loïc Magnan[2], Yves Robert[2]

[1] CSAIL, Massachusetts Institute of Technology, USA
[2] LIP, ENS Lyon, 46 Allée d'Italie, 69364 Lyon Cedex 07, France
UMR 5668 - CNRS - ENS Lyon - UCB Lyon - INRIA

kunal_ag@mit.edu,{Anne.Benoit|Loic.Magnan|Yves.Robert}@ens-lyon.fr

July 2009

LIP Research Report RR-2009-22

Abstract

Pipelined workflows are a popular programming paradigm for parallel applications. In these workflows, the computation is divided into several stages and these stages are connected to each other through first-in first-out channels. In order to execute these workflows on a parallel machine, we must first determine the *mapping* of the stages onto the various processors on the machine. After finding the mapping, we must compute the *schedule* — the order in which the various stages execute on their assigned processors.

In this paper, we explore the *scheduling problem* for linear workflows, assuming that the *mapping is given*. *Linear workflows* are a special case of workflows for which the dependencies between stages can be represented by a linear graph. The objective of the scheduling algorithm is either to maximize throughput or to minimize latency or both. We consider two realistic execution models: the one-port model and the multi-port model. In both models, finding a schedule to minimize latency is easy. However, computing the schedule to minimize period (maximize throughput) is NP-hard in the one-port model, but can be done in polynomial time in the multi-port model. We also present an approximation algorithm to minimize period in the one-port model. Finally, the bi-criteria problem, which consists in finding a schedule respecting a given period and a given latency, is NP-hard in both models.

Key words: pipeline graphs, workflow, scheduling, mapping, period, latency, bi-criteria, complexity results

Contents

1 Introduction

Pipelined workflows are a popular paradigm for *streaming applications* like video and audio encoding and decoding, DSP applications, etc. [6, 14]. Streaming applications are becoming increasingly prevalent, and many languages [5, 8, 10, 11, 15] are being continually designed to support these applications. In these languages, the program is represented by a *workflow graph*, which consists in several *stages* connected to each other using *first-in-first-out channels*. In order to execute these workflows on a parallel machine, we must first determine the *mapping* of these stages on the various processors of the machine. After finding the mapping, we must compute the *schedule*, that is, the order in which the various stages execute on their assigned processors. Since data continually flows through these applications, the goal is often to decrease *period* and/or *latency*.

Like Subhlok and Vondran [12, 13], we explore the problem of linear workflows. Linear workflows are a special case of workflows where stages and communications can be represented by a linear graph. Subhlok and Vondran studied the problem of mapping linear graphs on homogeneous platforms, and their work has been extended for heterogeneous platforms in [1, 3].

Since finding the optimal mapping is often NP-hard [1], we explore the problem of scheduling linear workflows, *given the mapping*. As in [1], we consider two realistic models. The first model is the *one-port model without overlap* where each processor can either compute or receive an incoming communication or send an outgoing communication at any time-step. This model does a good job of representing single-threaded systems. The second model we consider is the *bounded-multiport model with overlap*, which allows multiple incoming and outgoing communications simultaneously, and allows the processor to perform computation and communication at the same time. To the best of our knowledge, the problem of computing an optimal schedule for a given mapping has not been explored for linear workflows.

After giving some details about the framework (Section 2), we recall some related work (Section 3). Section 4 is devoted to the one-port model without overlap, and shows that finding a schedule with optimal latency has polynomial complexity, whereas finding a schedule with optimal period is NP-hard. The section also presents a 4-approximation algorithm for minimizing the period. Section 5 is devoted to the multi-port model with overlap, and shows that both finding a schedule with optimal latency and finding a schedule with optimal period can be done in polynomial time. However, finding a schedule which respects both a given period and a given latency is NP-hard.

2 Framework

This section is devoted to a precise statement of the different models and optimization problems. First, we explain the representation of programs by linear graphs. We then explain the parallel architecture of the machines and give two main models of computation, both realistic for different machines. Finally, we introduce important optimization problems.

2.1 Representation of a program by a linear graph

A workflow graph contains several stages, connected to each other by first-in-first-out (FIFO for short) channels. A lot of work has been done in the very general case, and many problems have

been shown to be NP-hard [2, 14, 16, 17]. Because of this, we only consider simple workflow applications whose graphs are linear chains. Such an application is represented in Figure 1.

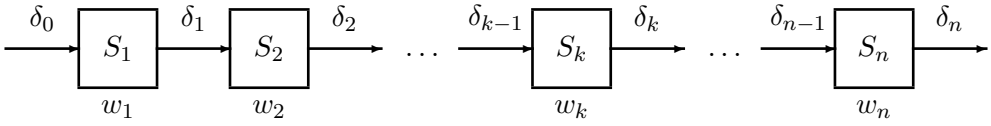


Figure 1: Representation of an application workflow by a linear graph. Stage S_k has a computation of size w_k, an incoming communication of size δ_{k-1} and an outgoing communication of size δ_k.

We assume that our pipeline graph consists of n stages S_k with $k \in [\![1, n]\!]$. A stage S_k receives an incoming communication of size δ_{k-1} from the previous stage, performs w_k computations and send a data item of size δ_k to the next stage. If computations and communications are done in parallel, the input for data set $i+1$ can be received while computing for data set i and sending result for data set $i-1$. Else, these tasks have to be done serially. We will deal with both models, with and without communication/computation overlap.

2.2 Platform

We assume that the platform consists of p processors P_u with $u \in [\![1, p]\!]$, which are fully interconnected as a virtual clique. There is a bidirectional link $link_{u,v} : P_u \leftrightarrow P_v$ between each processor pair (P_u, P_v)[1], and the corresponding bandwidth is $b_{u,v}$. It takes $X/b_{u,v}$ time-units to send a message of size X from P_u to P_v. The speed of processor P_u is denoted by s_u and it takes X/s_u times-units to execute X floating point operations on P_u. Such a platform, with 4 processors, is shown in Figure 2. The bounded capacity of each processor's network card is represented as follows: B_u^i (resp. B_u^o) represents the capacity of the input (resp. output) network card of processor P_u. P_u cannot receive more than $1/B_u^i$ data items per time-unit, and cannot send more than $1/B_u^o$ data items per time-unit.

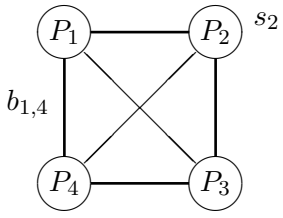


Figure 2: A platform with 4 processors. Processor P_2 has a speed s_2 and the bandwidth of the bidirectional link between P_1 and P_4 is $b_{1,4}$.

A platform is said to be:

- *fully homogeneous* if all processors are identical, that is, they compute at the same speed s, and all communication devices are identical, that is, the bandwidth is the same (say b) between any pair of processor and all network cards have the same capacities (say B^i and B^o).

[1] We suppose that $P_u \neq P_v$.

- *communication homogeneous* if communication devices are identical, that is, the bandwidth is the same (say b) between any pair of processor and all network cards have the same capacities (say B^i and B^o). However, in these platforms, processor speeds may differ ($s_u \neq s_v$).
- *fully heterogeneous* if there is *a priori* no relation between processors and between communications devices. This is the most general case, where two processors P_u and P_v may have different speeds ($s_u \neq s_v$), where two different links $link_{u,v}$ and $link_{u',v'}$ may have different bandwidths ($b_{u,v} \neq b_{u',v'}$), and where two network cards may have different limitations ($B^i_u \neq B^i_v$ and/or $B^o_u \neq B^o_v$).

Finally, we assume that two processors P_{in} and P_{out} are devoted to input/output data.

These platform models are all realistic. In general, multi-core processors can be represented by *fully homogeneous* platforms, whereas department clusters and large-scale grids are respectively *communication homogeneous* and *fully heterogeneous* platforms.

2.3 Model of computation

We now provide a precise description of the model of execution, since the results depend greatly on this model. There are basically two degrees of freedom in the model of execution:

- Can a processor send (resp. receive) some data to (resp. from) some different processors at the same time? The model is said to be *multi-port* if the answer is "yes", and *one-port* otherwise.
- Can communication and computation *overlap*, i.e. can a processor compute for a data set and send (resp. receive) communication for another data set at the same time?

This leads to four theoretical models. In fact, some of them are not realistic, and in this paper, we only consider *one-port without overlap* and *multi-port with overlap*, both of them without *preemption*: we cannot stop and restart later a communication or a computation:

- *One-port model without overlap*: This model does a good job for representing single-threaded systems. The complexity of finding the optimal mapping for this model has already been studied in [1, 3].
- *Multi-port model with overlap* This model is representative of current multi-threaded systems. Once again, the complexity of finding the optimal mapping for the model has already been studied in [1].

  It is important to point out that in the *multi-port model with overlap*, we require that communications use a constant bandwidth: having a variable bandwidth is reasonable when we allow preemption, which is not the case here. This means that when a communication of size δ_i begins between two different processors P_u and P_v, it uses a constant bandwidth $b_i \leqslant b_{u,v}$ for a time $t = \delta_i / b_i$. Such communications are represented in Figure 3.

2.4 Mapping

To execute the workflow linear graph, we have to assign application stages S_k to platform processors P_u. This assignment is called *a mapping* and the following section is devoted to a precise statement of this function.

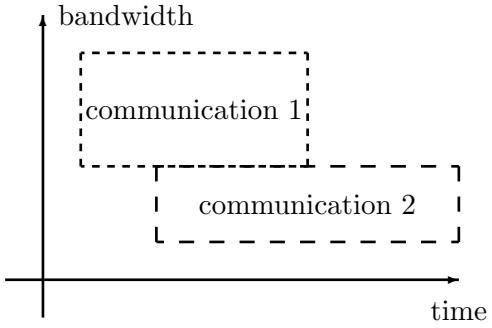


Figure 3: Representation of bandwidths used by two communications in the multi-port model.

2.4.1 Mapping representation

We define a function a:

$$a : [\![1, n]\!] \longrightarrow [\![1, p]\!] \tag{1}$$

This function is called *mapping* and makes the link between stages and processors. Typically, $a(k) = u$ means that the computation of a stage S_k is executed by processor $P_u = P_{a(k)}$. We extend the domain of definition of a to $\{0, \ldots, n+1\}$ by $a(0) = in$ and $a(n+1) = out$. This means $P_{a(0)} = P_{in}$ and $P_{a(n+1)} = P_{out}$, with P_{in} the input processor, and P_{out} the output processor, and it is used in many equations in this paper.

To represent a linear graph and a mapping on the same figure, we add over each stage S_k the corresponding processor $P_{a(k)}$. An example of this representation is given in Figure 4, where stages S_1 and S_3 are mapped on P_2 whereas stage S_2 is mapped on P_3 and stage S_4 on P_1.

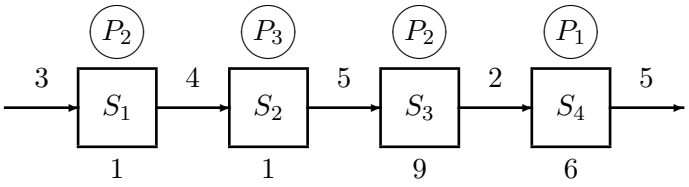


Figure 4: Representation of a linear graph and a mapping.

2.4.2 Cycle-time

Given a linear graph G and a mapping a, we define the working time of a processor P_u for the computation of each data set in steady-state mode (after the initialization phase). This quantity is called the *cycle-time* of processor P_u and is represented by $CT_u(a, G)$. When there is no confusion possible, we may omit a and G and refer to the cycle-time of P_u as CT_u. The formal definition of the cycle-time differs if we are in the one-port model or in the multi-port model, because communications and computations can be done in parallel in one model and not in the other one.

Formally, in the *one-port model*, the cycle-time of processor P_u is the sum of computation

times, input communication times and output communication times:

$$CT_u(a,G) = \sum_{j=1}^{n} \frac{w_j \mathbb{1}_{a(j)=u}}{s_u} + \sum_{j=1}^{n} \frac{\delta_{j-1} \mathbb{1}_{a(j)=u} \mathbb{1}_{a(j-1)\neq u}}{\min\left\{b_{a(j-1),u}, B_u^i\right\}} + \sum_{j=1}^{n} \frac{\delta_j \mathbb{1}_{a(j)=u} \mathbb{1}_{a(j+1)\neq u}}{\min\left\{b_{u,a(j+1)}, B_u^o\right\}} \tag{2}$$

and in the *multi-port model*, the cycle-time of processor P_u is the minimum time that a processor needs to accomplish all tasks for one kind of limitation (network card limitations, processor maximal speed, bandwidths limitations):

$$CT_u(a,G) = \max \begin{cases} \dfrac{1}{s_u} \displaystyle\sum_{a(i)=u} w_i \\ \displaystyle\max_{k\in\{1,\ldots,p\}\cup\{in,out\},k\neq u} \left\{ \dfrac{1}{b_{k,u}} \displaystyle\sum_{a(i)=k,a(i+1)=u} \delta_i \right\} \\ \displaystyle\max_{k\in\{1,\ldots,p\}\cup\{in,out\},k\neq u} \left\{ \dfrac{1}{b_{u,k}} \displaystyle\sum_{a(i)=u,a(i+1)=k} \delta_i \right\} \\ \dfrac{1}{B_u^o} \displaystyle\sum_{a(i)=u,\ a(i+1)\neq u} \delta_i \\ \dfrac{1}{B_u^i} \displaystyle\sum_{a(i-1)\neq u,\ a(i)=u} \delta_{i-1} \end{cases} \tag{3}$$

In both models, we call the cycle-time $CT(a,G)$ of a linear graph and a mapping the maximum processor cycle-time:

$$CT(a,G) = \max\left\{CT_1(a,G), \ldots, CT_p(a,G)\right\} \tag{4}$$

Once again, we may omit a and G when it is clear from context.

2.5 Schedule

We assume that the input data arrives periodically, every K time-units. In addition, we focus on periodic executions, where computations and communications are also of period K. For a given mapping, there exist different ways to execute the application on the platform. For example, suppose that the linear graph and the mapping are represented by Figure 5. The platform

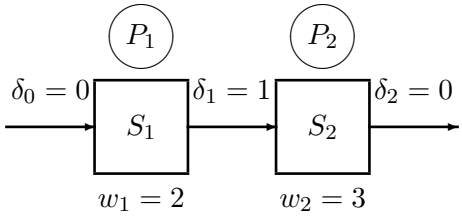


Figure 5: A simple linear graph.

is homogeneous and consists of two processors P_1 and P_2. The common computation speed is $s = 1$ and the limitations for communications are $b = B^i = B^o = 1$.

On Figure 6 are represented two different schedules of period $K = 4$, in the one-port model, when communications and computations occur at full speed. For each schedule, the computa-

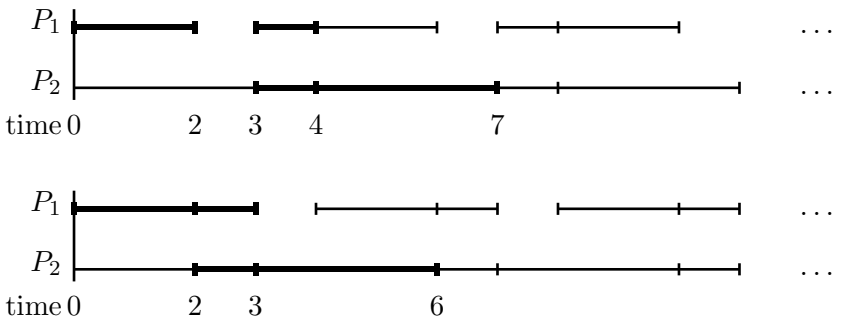


Figure 6: Two different schedules of same period $K = 4$ for a given linear graph, a given platform and a given mapping. For each schedule, the computation of one data set is represented in bold.

tion of one data set is drawn in bold. Computations of other data sets are also represented to show that the schedule has a period of 4. Notice that it takes 7 time-units to run the linear graph (i.e. to execute a data set entirely) in the first schedule and only 6 is the second one.

In both models, a schedule is given by the computation of one data set and by the period K, which permits to deduce the computations of all data-sets from the computation of one of them:

- *One-port model*: In the one-port model, a processor cannot send/receive data to/from two processors simultaneously. Because of this, it makes no sense to consider that a communication uses only a part of the bandwidth: the non-used bandwidth cannot be use for any other communication ! Therefore, we assume that, in the one-port model, every communication occurs using the maximal bandwidth which respects link bandwidths and network cards limitations. Formally, a communication from P_u to P_v uses a bandwidth b which follows
$$b = \min\left\{b_{u,v}, B_v^i, B_u^o\right\} \tag{5}$$
Similarly, we assume that a computation on P_u is done at maximal speed $s = s_u$.
- *Multi-port model*: In the multi-port model, it makes sense to have a communication which uses only a fraction of the available bandwidth, because different communications can occur at the same time. On the contrary, we assume that two computations cannot occur simultaneously on a processor P_u. As in the one-port model, we assume that a computation on processor P_u always occurs at full speed s_u.

2.5.1 Period K and cycle-time $CT(a, G)$.

It is easy to see that, for both models, the period K of any schedule has to be larger than the cycle-time $CT(a, G)$, which means that, for a given linear graph G, a given platform and a given mapping a, the period K of any schedule satisfies
$$CT(a, G) \leqslant K \tag{6}$$
This inequality states that the period has to be larger than the maximal amount of time necessary for any processor to execute computations and communications for one data set. Since an important part of this paper concerns period minimization, this observation is important: it gives a lower bound of the period K that is easy to compute. In proofs, theorems and experiments, we often compare the period K (and the optimal period $K_{\min}$) to the cycle-time CT.

2.6 Goal: minimize period and/or latency

There are basically three important objectives in parallel executions:

- to minimize the period K (inverse of throughput) i.e. finding $K_{\min}$
- to minimize the latency L (response time) i.e. finding $L_{\min}$
- to enforce a given period and a given latency (bi-criteria)

We point out that these two optimization criteria are anagonistic to each other.

In this paper, we always consider that the linear graph and platform are given. In the related work section (Section 3), we recall that the problems of finding a mapping that minimizes period, latency or by-criteria are often NP-hard. Therefore, for this paper, we assume that the mapping is given, and we address the complexity of finding a schedule that minimizes period or latency. More exactly, we show in Sections 4 and 5 that:

- finding the optimal latency $L_{\min}$ is easy in both computation models.
- finding the optimal period $K_{\min}$ is NP-hard in the one-port model without overlap, but has polynomial complexity in the multi-port model with overlap.
- respecting a given period K and a given latency L is easily proved to be NP-hard in the one-port model without overlap[2] and is also NP-hard in the multi-port model with overlap: this second result is interesting and surprising because period minimization and latency minimization are both polynomial for this model.

2.7 Example

To explain the previous notations, we explore a basic example, represented on Figure 7. We

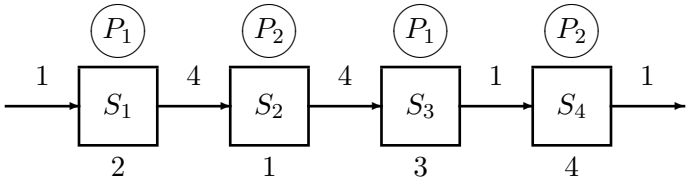


Figure 7: A simple example of linear graph.

assume that the platform is fully-homogeneous, and consists of two processors P_1 and P_2 of speed $s = 1$. Network card capacities are $B^i = B^o = 1$, and links have same bandwidth $b = 1$. The goal is to minimize the period.

Stages S_1 and S_3 are mapped on P_1 whereas stages S_2 and S_4 are mapped on P_2. Such a mapping balances computations. The computation of a single data set on this graph can be represented by the schedule in Figure 8 and the next array. This computation fits both the one-port model and the multi-port one.

[2]since finding the optimal period is already NP-hard.

$in \to P_1$	1																					
P_1		2	3											13	14	15						
$P_1 \to P_2$				**4**	**5**	**6**	**7**										**16**					
P_2								8										17	18	19	20	
$P_2 \to P_1$									**9**	**10**	**11**	**12**										
$P_2 \to out$																						21

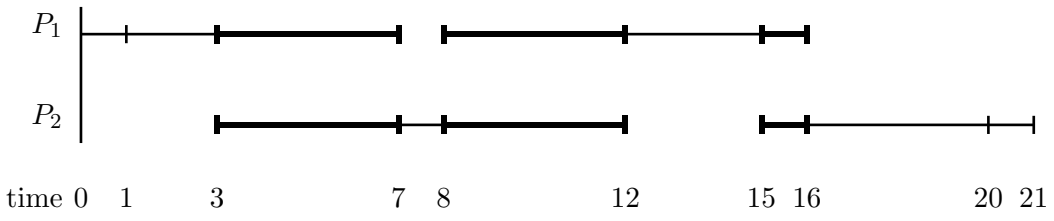


Figure 8: Computation of a single data set on the graph represented in Figure 7. Communications between P_1 and P_2 are represented in bold whereas computations on P_1 or on P_2 and input and output communications are not.

We now assume that we are in the multi-port model with overlap (computations/communications can overlap and a processor can have simultaneously different incoming/outgoing communications).

Obviously, choosing a period of 21 for the schedule represented in Figure 8 respects all constraints, because 21 is also the latency of this schedule. We can now try to find a better period for this schedule, or search for other schedules with smaller periods.

By definition of the cycle-time (see Equation 3), we have for processor P_1:

$$CT_1 = \max\left\{\frac{w_1+w_3}{s}, \frac{\delta_0+\delta_2}{b}, \frac{\delta_1+\delta_3}{b}, \frac{\delta_1+\delta_3}{B^o}, \frac{\delta_0+\delta_2}{B^i}\right\} = 5$$

and similarly, the cycle-time of processor P_2 follows

$$CT_2 = \max\left\{\frac{w_2+w_4}{s}, \frac{\delta_1+\delta_3}{b}, \frac{\delta_2+\delta_4}{b}, \frac{\delta_2+\delta_4}{B^o}, \frac{\delta_1+\delta_3}{B^i}\right\} = 5$$

The cycle-time of the linear graph and the mapping CT follows

$$CT = \max\{CT_1, CT_2\} = 5$$

The period of any schedule for this linear graph and this mapping is bigger than the cycle-time (see equation 6), which means that, for any schedule of period K,

$$K \geqslant 5$$

and the optimal period $K_{\min}$ for this linear graph and this mapping follows

$$K_{\min} \geqslant 5$$

The schedule given by Figure 8 cannot be executed with a period of 5. For example, incoming communications on P_1, represented in bold Figure 9, cannot be executed every 5 time units: the communication from P_2 to P_1 for data set i (upper schedule) would occur at the same time than the input communication for data set $i-2$ (lower schedule), and reach the network card limitation $B^i = 1$ in the interval of time $[10, 11]$.

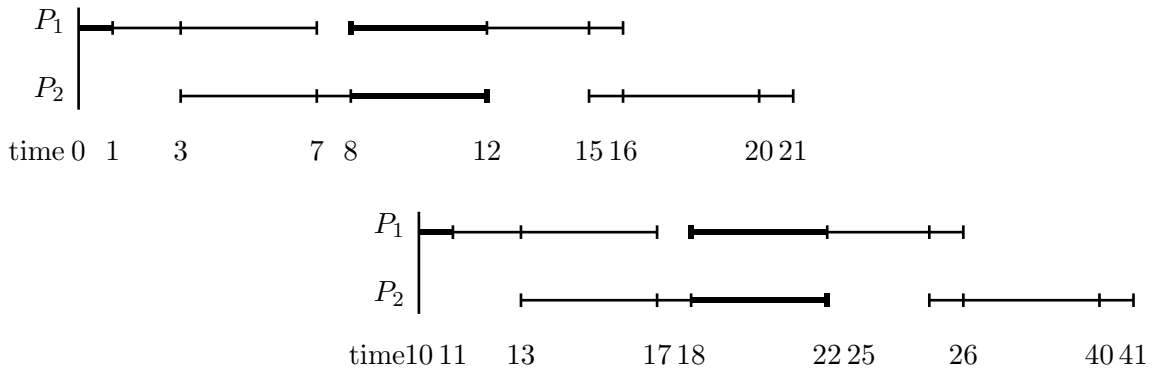


Figure 9: Computation of two data sets on the graph represented in Figure 7. The upper schedule represents the computation of one data set i, whereas the lower one represents the computation of the data set $i-2$, when the period is supposed to be 5. In bold are represented incoming communications of processor P_1, and cannot be computed in parallel for different data sets without reaching the network card limitation B^i. Because of the conflict in the interval of time $[10, 11]$, here is a counter example: such a schedule is not admissible.

For this graph and this mapping, it is possible to build a schedule with period 5. Such a schedule is represented in Figure 10. Ones can verifies that with a period of 5, all processors speeds, all bandwidths and all network card limitations are respected. This schedule has an optimal period $K = K_{\min} = 5$, but a latency of 31 which is bigger than the optimal latency $L_{\min} = 21$.

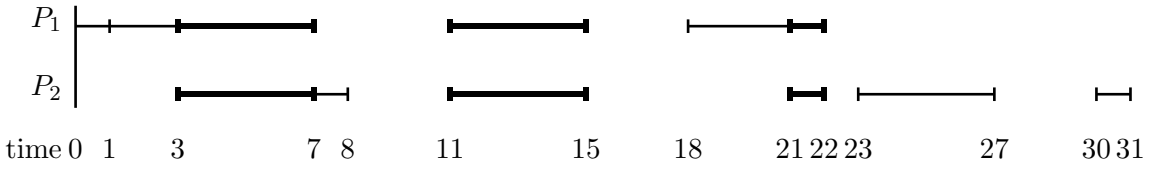


Figure 10: A schedule of optimal period $K = K_{\min} = 5$ for the graph and the mapping represented in Figure 7. Computations on P_1 and on P_2 are represented in bold whereas communications between P_1 and P_2 and input and output communications are not.

Finally, for this example, the optimal period $K_{\min}$ is equal to the cycle-time CT. Later in this paper, we prove this result holds true for any graph and any mapping in the multi-port model (see Theorem 5.2).

3 Related work

A lot of work has been done in the problem of scheduling workflows. This section is devoted both to models used for representing workflows and/or platforms, and to complexity results, especially those concerning the mapping.

3.1 Model for the application

Many languages [5, 8, 10, 11, 15] are being continually designed to support streaming applications. In these languages, a program is represented by a **workflow graph**, which consists of *stages*. Two stages can be connected to each other and the *link* between these stages is a bidirectional first-in-first-out channel. Moreover, there are specific links for the input and the output.

Many papers focus on **workflows represented by Directed Acyclic Graphs** (DAGs for short). Since data continually flows through streaming applications, the goal of a scheduler is often to decrease the *period* and/or the *latency* [6, 14, 16, 17]. We saw in the previous section that this is done by choosing both a mapping and a schedule. In the very general case where the workflow graph is not constrained, most problems related to period and/or latency minimization are NP-hard. Because of this, many simplifications have been considered for this problem. Vydynathan et al. consider bi-criteria minimization problems on homogeneous platforms. In [16], they explain that finding the optimal mapping is NP-hard and they give a mapping and scheduling heuristic for applications workflows represented as DAGs. Their algorithm minimizes the latency of workflows while satisfying strict throughput requirements. In [17], they also describe a heuristic that optimizes throughput of streaming workflows while meeting latency constraints. These papers provide many interesting ideas and several heuristics to solve the general mapping problem. Similarly, Taura and Chien [14] and Beaumont et al. [2] consider applications represented with DAGs. In both cases, the problem of finding an optimal mapping that minimizes period on heterogeneous platforms is NP-hard, but heuristics are given.

A lot of work has been done with a stronger simplification, where **workflows are represented using linear graphs**. Subhlok and Vondran studied the problem of mapping linear graphs on homogeneous platforms. In [12], they prove that minimizing period is polynomial and has a time complexity $O(P^4k^2)$ when P is the number of processors and k the number of tasks. In [13], they extend their results and show that finding the optimal latency respecting a period has also a time complexity $O(P^4k^2)$. We point out that they assume that the number of processors P is larger than the number of tasks k, which is a strong assumption that we do not use in this paper. Without this assumption, minimizing period is easily proved to be NP-hard, using a reduction to 2-PARTITION [7].

3.2 Model for the platform

The standard model for DAG scheduling heuristics does a poor job to model physical limits of interconnection networks. The model assumes an unlimited number of simultaneous sends and receives, i.e. a network card of infinite capacity, on each processor. A more realistic model is the **one-port model** [4], where a processor can be involved in a single communication, either a send or a receive. Obviously, independent communications between distinct processor pairs can take place simultaneously. Some work has already been done, where workflows are *linear* and under the *one-port model* [1, 3]: the problem of finding a mapping that minimizes period is proved to be NP-hard for one-to-one mappings, interval-based mappings and general mappings.

Because of this, the bi-criteria problem is also NP-hard on heterogeneous plaftorms.

An other realistic model we use in this paper is the **bounded multiport model** [9]. This model allows multiple incoming and outgoing communications but the total communication incoming (resp. outgoing) volume is bounded by the capacity of the network card. Once again, some work has already been done, where workflows are *linear* and under the *bounded multiport model* [1]. It is proved that finding the mapping that minimizes period is NP-hard on fully homogeneous platforms[3]. Moreover, finding the mapping that minimizes latency is proved to be NP-hard on communication homogeneous platforms and on fully heterogeneous platforms.

[3]This means that this problem is also NP-hard on communication homogeneous platforms and on fully heterogeneous platforms.

4 Finding the optimal schedule for a given mapping in the one-port model

In this part, we explore the problem of period and/or latency minimization in the one-port model without overlap. We first deal with latency minimization, then comes period minimization and finally the bi-criteria problem.

4.1 Latency

Theorem 4.1. *Given a linear workflow and a mapping, the problem of computing the schedule that leads to the optimal latency has polynomial complexity in the one-port model without overlap.*

Proof. For minimizing latency, we can consider schedules with periods long enough to separate the computation of different data sets in the linear workflow. This way, the optimal order is obvious: we execute all the computations and all the communications as soon as possible. The corresponding latency is the sum of all computation and communication times. □

4.2 Period

4.2.1 Finding the optimal period is NP-hard

Theorem 4.2. *Given a linear workflow and a mapping, the problem of computing the schedule that leads to the optimal period is NP-hard in the one-port model without overlap.*

Proof. We consider the corresponding decision problem and show that it is NP-complete: given a linear graph, a mapping, and a bound B, does there exist a schedule such that the period does not exceed B? This problem is obviously in NP: given a linear graph, a mapping, and a schedule, we can easily compute the period K and check whether it does not exceed B. To establish the completeness, we use a reduction from 2-PARTITION [7], which is NP-complete in the weak sense. We consider an instance $\mathcal{I}_1$ of this problem: given a list of integers $(a_i)_{1 \leqslant i \leqslant n}$ such that $\sum_{i=1}^{n} a_i = B$, does there exist $\gamma \in \{1, \ldots, n\}$ such that

$$\sum_{i \in \gamma} a_i = \sum_{i \notin \gamma} a_i = B/2 \tag{7}$$

We associate to $\mathcal{I}_1$ an instance $\mathcal{I}_2$ with $2n + 2$ processors, given by the linear graph and the mapping represented on Figure 11. The size of $\mathcal{I}_2$ is obviously linear in the size of $\mathcal{I}_1$. We now show that $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_2$ has one, i.e. if and only if there exists a schedule of period smaller than B for the graph and the mapping given by Figure 11. The corresponding execution graph is given by Figure 12.

Intuitively, we note that processor P_0 has many successors and P_{2n+1} many predecessors (see Figure 12). We need the ordering of the associated communications to compute the optimal period for this execution graph. Because we deal with the period, we suppose that two different processors can run in parallel, which is true if they deal with different data sets.

Suppose first that $\mathcal{I}_1$ has a solution γ. We compute the following operation list for $\mathcal{I}_2$: P_0 first communicates with $P_1, P_3, \ldots, P_{2n-1}$. All of this is done at the beginning of a period, for different data sets, because these communications are free. Then, P_0 makes his computation which lasts B time units. The schedule of P_0 is represented in Figure 13a.

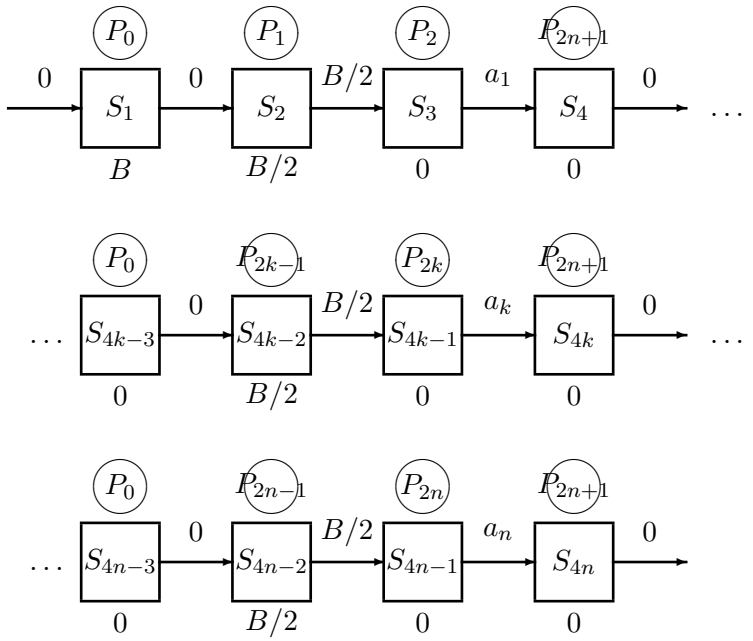


Figure 11: A linear graph and a mapping to prove that minimizing period in the one-port model is NP-hard.

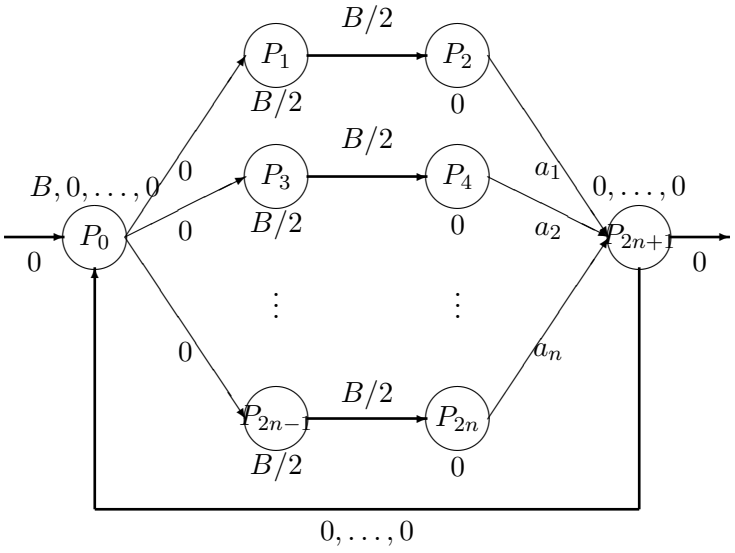


Figure 12: The execution graph corresponding to the linear graph and the mapping given in Figure 11.

Then, we do the computations of processors $(P_{2i-1})_{i\in\gamma}$ in the interval of time $[0, B/2]$ and the communications between $(P_{2i-1}, P_{2i})_{i\in\gamma}$ in the interval of time $[B/2, B]$ (see Figure 13b). On the contrary, we do the communications between $(P_{2i-1}, P_{2i})_{i\notin\gamma}$ in the interval of time $[0, B/2]$ and then the computations of processors $(P_{2i-1})_{i\notin\gamma}$ in the interval of time $[B/2, B]$(see Figure 13c). We also do the communications between $(P_{2i}, P_{2n+1})_{i\in\gamma}$ one per one in an unspecified order in the interval of time $[0, B/2]$: this is possible because $\sum_{i\in\gamma} a_i = B/2$ by hypothesis (see Figure 13d). Similarly, we do the communications between $(P_{2i}, P_{2n+1})_{i\notin\gamma}$ one per one in an unspecified order in the interval of time $[B/2, B]$ (see Figure 13e).

We obtain a schedule of period B, which ends the first part of the proof. Ones can notice that this construction is possible because we have $\sum_{i\in\gamma} a_i = \sum_{i\notin\gamma} a_i = B/2$, so we can "2-PARTITION" communications with P_{2n+1} and avoid to have a communication which starts before $B/2$ and ends after $B/2$ (see Figure 13f).

We now prove that any schedule of period $K \leqslant B$ "looks like" the previous schedule and "2-PARTITION" communications with P_{2n+1}. Let us suppose that $\mathcal{I}_2$ has a solution. There is a schedule of period $K \leqslant B$. Since there is a computation of size B on P_0, we know that $K \geqslant B$ and finally $K = B$. Processor P_0 computes for a data set k_0 between t_0 and $t_0 + B$. There is no idle time for P_0, so for all $1 \leqslant i \leqslant n$, there is a data set such that the communication between P_0 and P_{2i-1} is done at $t = t_0$. For all $1 \leqslant i \leqslant n$, there is no idle time for processor P_{2i-1}, so we have either a computation on P_{2i-1} for $t \in [t_0, t_0 + B/2]$ and a communication between P_{2i-1} and P_{2i} for $t \in [t_0 + B/2, t_0 + B]$, either a communication between P_{2i-1} and P_{2i} followed by a computation on P_{2i-1}. We define γ as follows:

$$\gamma = \{1 \leqslant i \leqslant n, \text{ there is a computation on } P_{2i-1} \text{ for } t \in [t_0, t_0 + B/2]\} \tag{8}$$

Let j be in γ. By hypothesis, there is a computation on P_{2j-1} for $t \in [t_0, t_0 + B/2]$. There is no idle time for P_{2j-1}, so there is a communication between P_{2j-1} and P_{2j} for $t \in [t_0 + B/2, t_0 + B]$. A communication between P_{2j} and P_{2n+1} occurs in the interval of time $t \in [t_0, t_0 + B/2]$. Because we cannot parallelize incoming communications of P_{2n+1}, communications between $P_{2i-1}, (i \in \gamma)$ and P_{2n+1} are done during independent intervals of time included in $[t_0, t_0 + B/2]$, so we have:

$$\sum_{i\in\gamma} a_i \leqslant B/2 \tag{9}$$

The same way, we can show that communications between $P_{2i-1}, i \notin \gamma$ and P_{2n+1} are done during independent intervals of time included in $[t_0 + B/2, t_0 + B]$, so we have

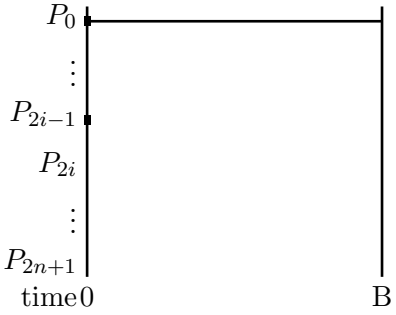
$$\sum_{i\notin\gamma} a_i \leqslant B/2 \tag{10}$$

By hypothesis, we have $\sum_{i\in\gamma} a_i + \sum_{i\notin\gamma} a_i = \sum_{1\leqslant i\leqslant n} a_i = B$, so $\sum_{i\in\gamma} a_i = \sum_{i\notin\gamma} a_i = B/2$. This ends the proof. □

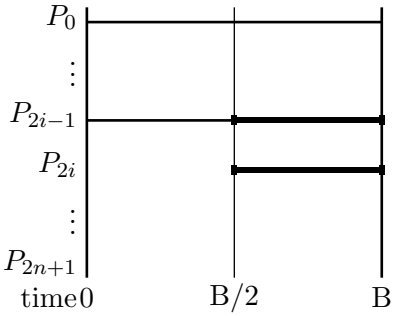
4.2.2 The greedy algorithm

The goal is to find a schedule with a small period in polynomial time in the one-port model without overlap when the mapping is given. By Theorem 4.2, we saw that finding the optimal period is NP-hard, therefore we now try to find some approximations.

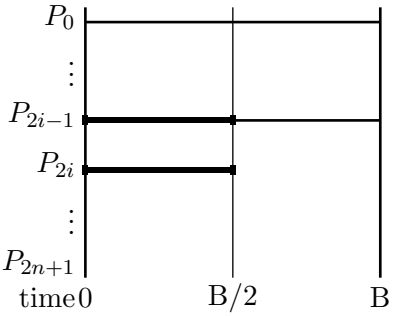
A first algorithm is to put each task (communication or computation) in the schedule as soon as possible. This algorithm is called *Greedy Algorithm*.



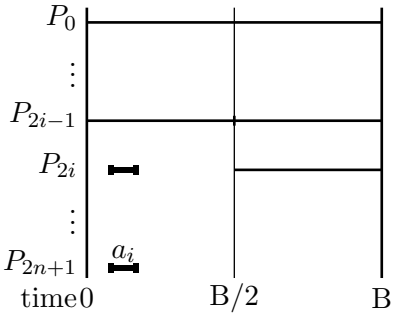
(a) Free **communications** and one computation on P_0



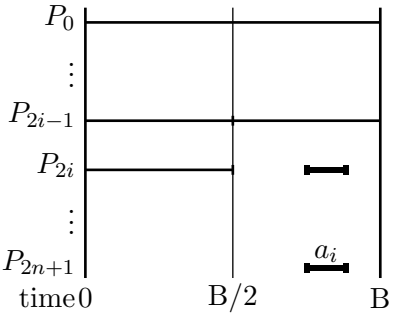
(b) If $i \in \gamma$, there is a computation for $t \in [0, B/2]$ on P_{2i-1} and a **communication** for $t \in [B/2, B]$ between P_{2i-1} and P_{2i}.



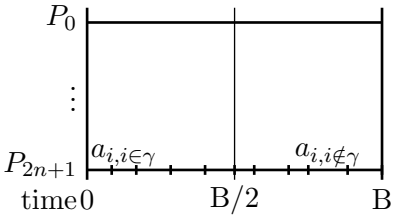
(c) If $i \notin \gamma$, there is a **communication** for $t \in [0, B/2]$ between P_{2i-1} and P_{2i} and a computation for $t \in [B/2, B]$ on P_{2i-1}.



(d) If $i \in \gamma$, the **communication** between P_{2i} and P_{2n+1} occurs in the interval of time $[0, B/2]$.



(e) If $i \notin \gamma$, the **communication** between P_{2i} and P_{2n+1} occurs in the interval of time $[B/2, B]$.



(f) This schedule uses the 2-PARTITION of $\{a_1, \ldots, a_n\}$.

Figure 13: Creation of a schedule of period B.

Algorithm 1 The Greedy Algorithm

for all tasks, communications and computations **do**
  add the task as soon as possible in the schedule
end for

We detail this algorithm on a basic example:

Let us suppose that the linear graph and the mapping are given by Figure 14.

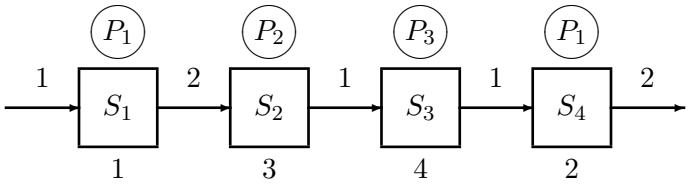


Figure 14: A linear graph and a mapping to explain the greedy algorithm.

We assume that the platform is fully-homogeneous and consists of $p = 3$ processors P_1, P_2 and P_3 of speeds $s = 1$. Network card capacities are $B^i = B^o = 1$ and links between these processors have bandwidths $b = 1$. The mapping is given: S_1 and S_4 are mapped on P_1 whereas S_2 and S_3 are respectively mapped on P_2 and P_3.

The list of tasks that the greedy algorithm has to add to the schedule consists of 5 communications and 4 computations. If we note $com(P_i, P_j, s)$ a communication from P_i to P_j of size s and $cp(P_k, t)$ a computation of size t on processor P_k, the list of tasks in this example is:

$$\{com(P_{in}, P_1, 1), cp(P_1, 1), \mathbf{com(P_1, P_2, 2)}, cp(P_2, 3), \mathbf{com(P_2, P_3, 1)}, cp(P_3, 4), \mathbf{com(P_3, P_1, 1)}, cp(P_1, 2), com(P_1, P_{out}, 2)\}$$

The algorithm firstly adds the communication $com(P_{in}, P_1, 1)$ to an empty schedule (see Figure 15). Then, it adds the computation $cp(P_1, 1)$, followed by the communication $\mathbf{com(P_1, P_2, 2)}$

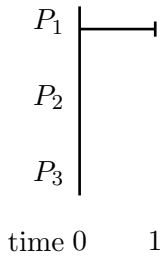


Figure 15: Construction of a schedule using the greedy algorithm: addition of the input communication on P_1.

and by the computation $cp(P_2, 3)$ (see Figure 16). At this point, the algorithm has to add the communication $\mathbf{com(P_2, P_3, 1)}$. In the schedule, processors P_2 and P_3 are not working at the

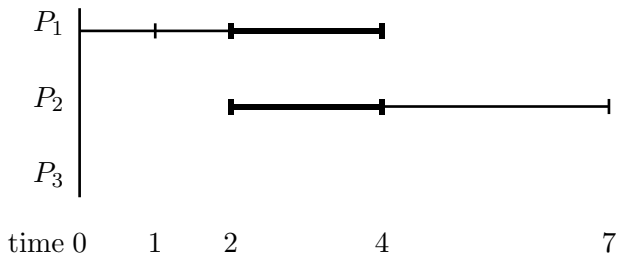


Figure 16: Construction of a schedule using the greedy algorithm.

first step of time. The algorithm puts the communication between these two processors as soon as possible, i.e. not at step of time 8 but at step of time 1 (see Figure 17). The final schedule is

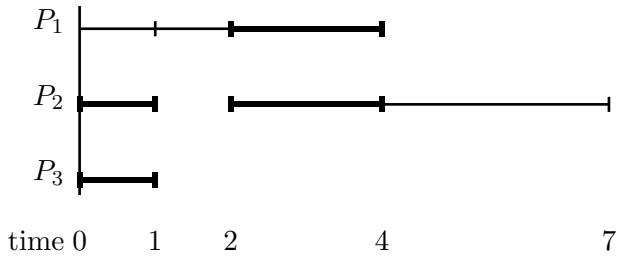


Figure 17: Construction of a schedule using the greedy algorithm.

represented in Figure 18. In this case, the algorithm returns a schedule with a period K of 10.

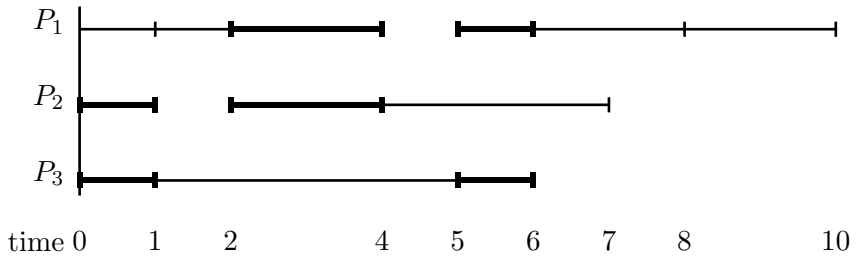


Figure 18: Construction of a schedule using the greedy algorithm.

In fact, the optimal period $K_{\min}$ is 9.

4.2.3 The communications first algorithm

All the constraints that we can find in the greedy algorithm come from the communications between processors: in the case where there are only computations, each processor can just compute its tasks in any order, and the period is the time needed to the busiest processor to compute all its tasks. This leads us to compute the schedule with the greedy algorithm slightly modified: we first use the greedy algorithm to add all the communications between processors (except P_{in} and P_{out}) in the schedule, and then we use it to add the computations.

Algorithm 2 The Communications First Algorithm

for all communications of the linear graph **do**
  add the communication as soon as possible in the schedule
end for
for all computations of the linear graph **do**
  add the computation as soon as possible in the schedule
end for

The schedule obtained by this algorithm running on the previous example (see Figure 14) for the communications is given by Figure 19 and the schedule for all the tasks by Figure 20.

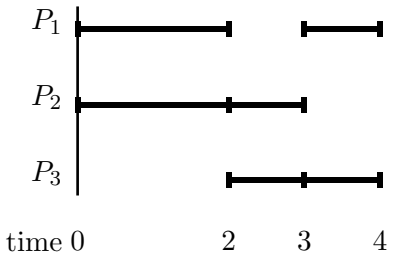


Figure 19: Construction of a schedule using the Communications First algorithm. First step: addition of communications.

On this example, the Communications First algorithm is strictly better than the Greedy one,

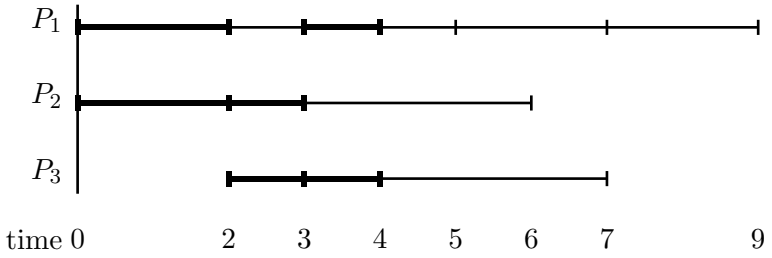


Figure 20: Construction of a schedule using the Communications First algorithm. Second step: addition of computations.

and computes a schedule of optimal period. In fact, some tests (detailed later in this paper) show that this new algorithm seems to be slightly better than the greedy one.

4.2.4 The longest first algorithm

It is reasonable to try to add tasks in the schedule from the biggest to the smallest, because if there is a gap in the schedule, a small task can be more easily placed than a big one. We first explain why we can say that a task is *bigger than an other one*[4], and then we give a new

[4]Typically, we can a priori not say than a task lasts longer than an other one if the mapping is not given because we have to take into account that there are different bandwidths, and processors may not compute at same speed.

algorithm to compute a schedule.

In the one-port model without overlap, we now exactly how long each task (communication or computation) will take. A computation of size ω_k on a processor $P_{a(k)}$ lasts $\omega_k/s_{a(k)}$ time units, and a communication of size δ_k from $P_{a(k)}$ to $P_{a(k+1)}$[5] lasts $\delta_k/\min(l_{a(k),a(k+1)}, B^o_{a(k)}, B^i_{a(k+1)})$ time units. This means that we always use the full processor speed for computations and the maximum bandwidth for communications.

Because of this, it makes sense to say that we take tasks (communications and computations) from the longest to the shortest. The new algorithm is simple: we add to a initially empty schedule all the tasks, from the longest to the shortest, as soon as possible in the schedule.

Algorithm 3 The Longest First Greedy Algorithm

for all tasks, communications and computations, *from the biggest one to the smallest one* **do**
  add the task as soon as possible in the schedule
end for

Since this algorithm is very similar to the Greedy algorithm (see algorithm 1), we do not run it on an example.

4.2.5 Experimentations on random graphs

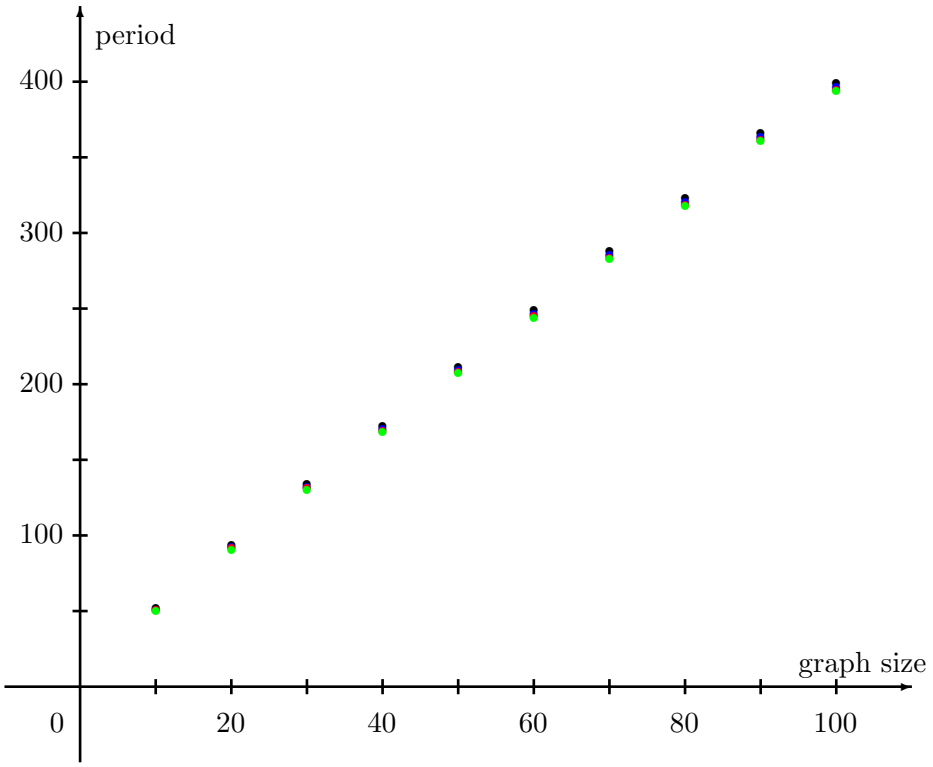
Given a size n, and four integers a, b, c, d such that $a \leqslant b$ and $c \leqslant d$, we can construct a random linear graph of size n by choosing the size of each communication $\delta_i,\ i \in [\![0, n]\!]$ uniformly in $[\![a, b]\!]$ and the size of each computation $w_j,\ j \in [\![1, n]\!]$ uniformly in $[\![c, d]\!]$. If we have a set of processors $\{P_1, P_2, \ldots, P_p\}$, we can construct a random mapping by assigning to each stage $S_k,\ k \in [\![1, n]\!]$ a random processor uniformly chosen in $\{P_1, P_2, \ldots, P_p\}$.

We give some average results, for random tests, when the platform is fully-homogeneous and consists of $p = 3$ processors, with same speed $s = 1$ and same bandwidth $b = 1$. Card limitations are $B^o = B^i = 1$. We took $(a, b) = (c, d) = (0, 9)$.

Size of linear graph	10	20	30	40	50	60
Number of tests	1000	950	900	850	800	750
Greedy algorithm	52.152	93.5779	133.806	172.526	211.405	248.741
Communications first algorithm	50.891	92.0905	131.933	170.433	209.44	246.656
Longest first algorithm	50.714	91.4863	131.132	169.342	207.954	244.78
Cycle-time	50.077	90.5989	130.158	168.46	207.15	244

Size of linear graph	70	80	90	100
Number of tests	700	650	600	550
Greedy algorithm	287.9	323.028	365.89	399.435
Communications first algorithm	285.431	320.534	363.408	396.72
Longest first algorithm	283.353	318.306	361.185	394.22
Cycle-time	282.594	317.602	360.592	393.705

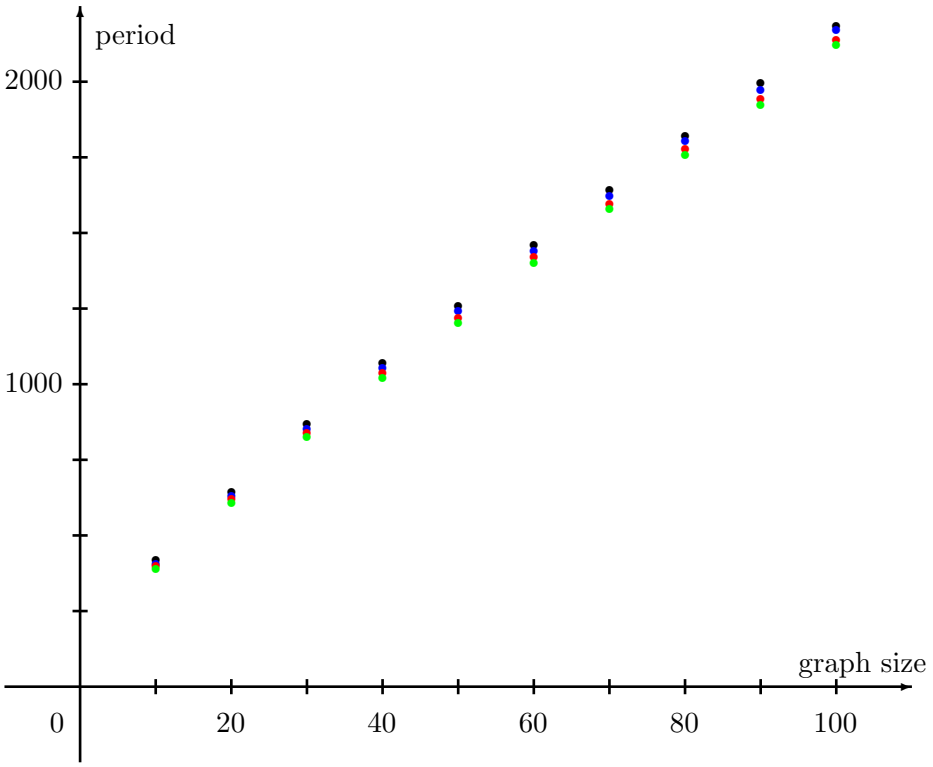
[5] $P_{a(k)} \neq P_{a(k+1)}$.



We also give some average results, for random tests, when the platform is fully-homogeneous and consists of $p = 10$ processors, with same speed $s = 1$ and same bandwidth $b = 1$. Card limitations are $B^o = B^i = 1$. We took $(a, b) = (c, d) = (0, 99)$.

Size of linear graph	10	20	30	40	50	60
Number of tests	1000	1000	1000	1000	1000	500
Greedy algorithm	417.267	644.373	867.466	1070.49	1258.05	1458.95
Communications first algorithm	405.223	629.444	851.974	1052.56	1240.58	1441.31
Longest first algorithm	398.541	619.962	837.387	1035.38	1217.29	1419.07
Cycle-time	390.085	607.363	823.857	1020.2	1201.01	1401.95

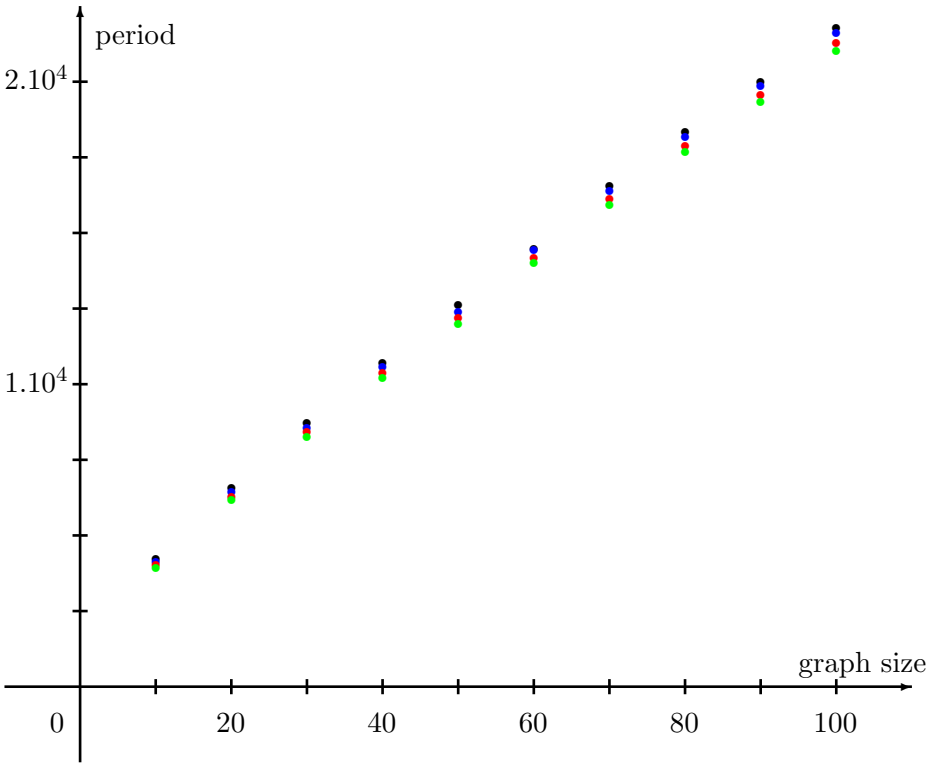
Size of linear graph	70	80	90	100
Number of tests	500	500	500	500
Greedy algorithm	1641.56	1820.72	1994.31	2185.77
Communications first algorithm	1623.29	1804.42	1973.07	2171.72
Longest first algorithm	1595.65	1778.33	1944.45	2138.3
Cycle-time	1578.17	1758.57	1924.07	2119.7



Some average results, for random tests, when the platform is fully-homogeneous and consists of $p = 10$ processors, with same speed $s = 1$ and same bandwidth $b = 1$. Card limitations are $B^o = B^i = 1$. We took $(a, b) = (c, d) = (0, 999)$.

Size of linear graph	10	20	30	40	50	60
Number of tests	1000	950	900	850	800	750
Greedy algorithm	4233.96	6565.5	8716.86	10736.5	12564.2	14595.7
Communications first algorithm	4113.6	6424.85	8560.47	10563.4	12376.8	14424.6
Longest first algorithm	4013.19	6284.16	8403.8	10376.5	12173.4	14171.6
Cycle-time	3918.17	6155.42	8249.46	10204.8	11986.1	13991.8

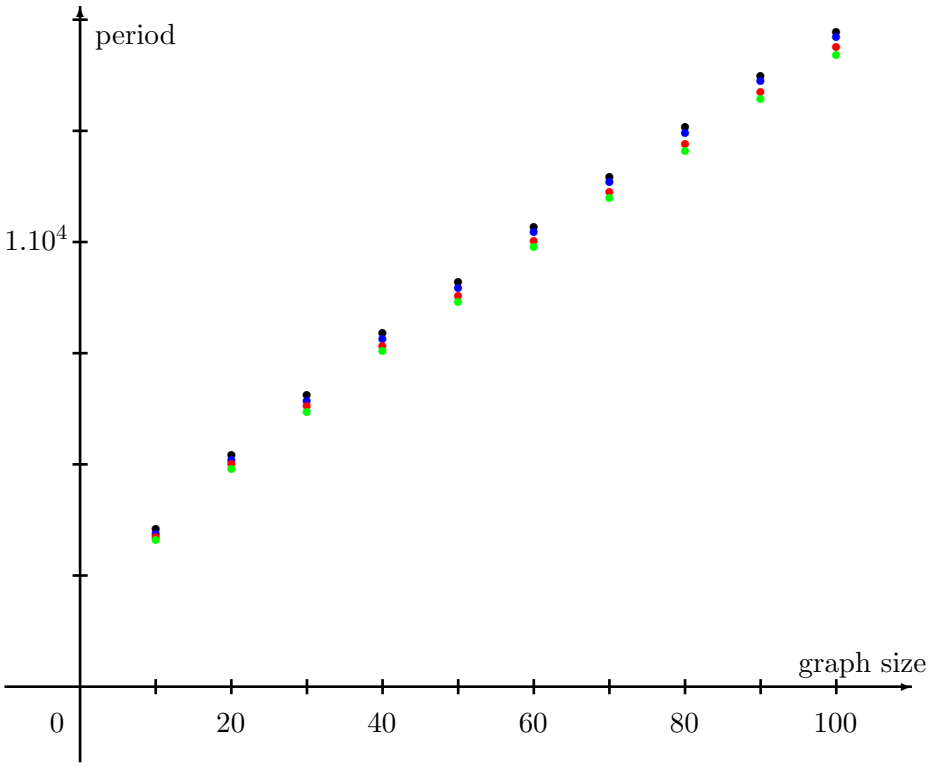
Size of linear graph	70	80	90	100
Number of tests	700	650	600	550
Greedy algorithm	16564.3	18327	19988.7	21772
Communications first algorithm	16375.8	18183.4	19865.4	21604.4
Longest first algorithm	16136	17890.9	19552.7	21265.1
Cycle-time	15913.1	17684.5	19342.3	21022.7



Some average results, for random tests, when the platform is fully-homogeneous and consists of $p = 20$ processors, with same speed $s = 1$ and same bandwidth $b = 1$. Card limitations are $B^o = B^i = 1$. We took $(a, b) = (c, d) = (0, 999)$.

Size of linear graph	10	20	30	40	50	60
Number of tests	1000	950	900	850	800	750
Greedy algorithm	3554.76	5217.48	6559.74	7953.39	9095.66	10325.8
Communications first algorithm	3426.55	5091.81	6425.09	7826.91	8960.36	10212.6
Longest first algorithm	3371.81	4997.29	6303.7	7668.32	8776.25	10011.7
Cycle-time	3290.37	4890.45	6183.69	7548.35	8638.13	9878.1

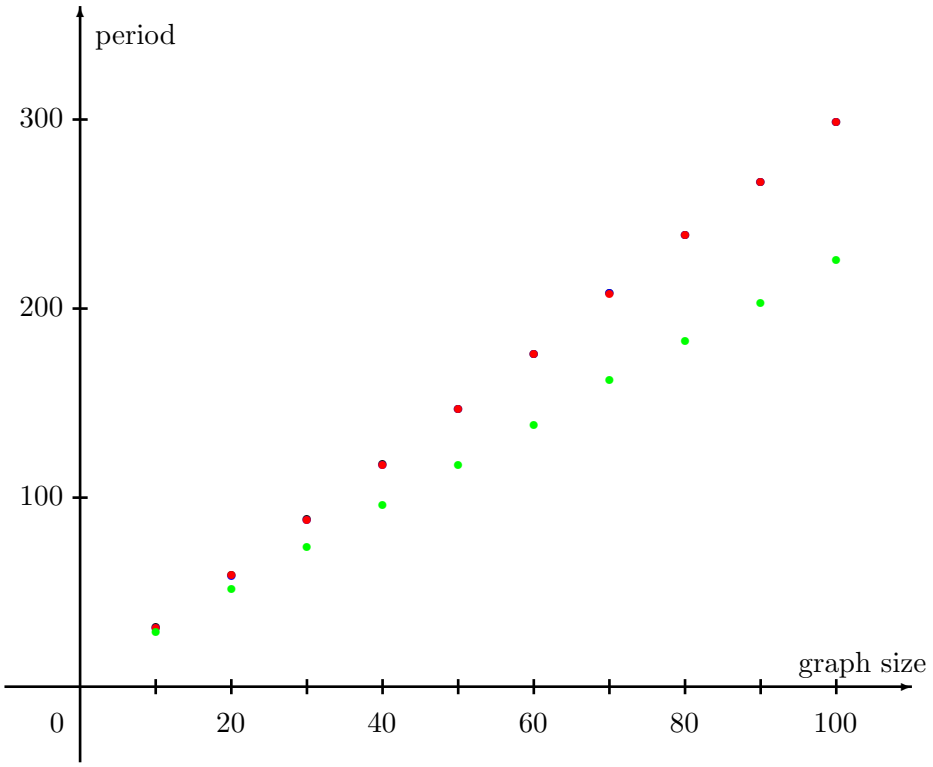
Size of linear graph	70	80	90	100
Number of tests	700	650	600	550
Greedy algorithm	11453.3	12580.3	13739.4	14719.2
Communications first algorithm	11335	12447	13615.2	14615.7
Longest first algorithm	11127.5	12208.3	13366.6	14376.3
Cycle-time	10976.5	12049.1	13220.5	14207.6



Some average results, for random tests, when the platform is fully-homogeneous and consists of $p = 3$ processors, with same speed $s = 1$ and same bandwidth $b = 1$. Card limitations are $B^o = B^i = 1$. We took $(a, b) = (0, 9)$ and $(c, d) = (0, 0)$, which means that all computation's size are 0.

Size of linear graph	10	20	30	40	50	60
Number of tests	1000	950	900	850	800	750
Greedy algorithm	31.362	59.1084	88.44	117.696	147.143	176.107
Communications first algorithm	30.915	58.6811	88.1167	117.278	146.721	175.787
Longest first algorithm	30.811	58.8158	88.2578	117.362	146.84	175.921
Cycle-time	28.873	51.6253	74.0811	96.1412	117.353	138.156

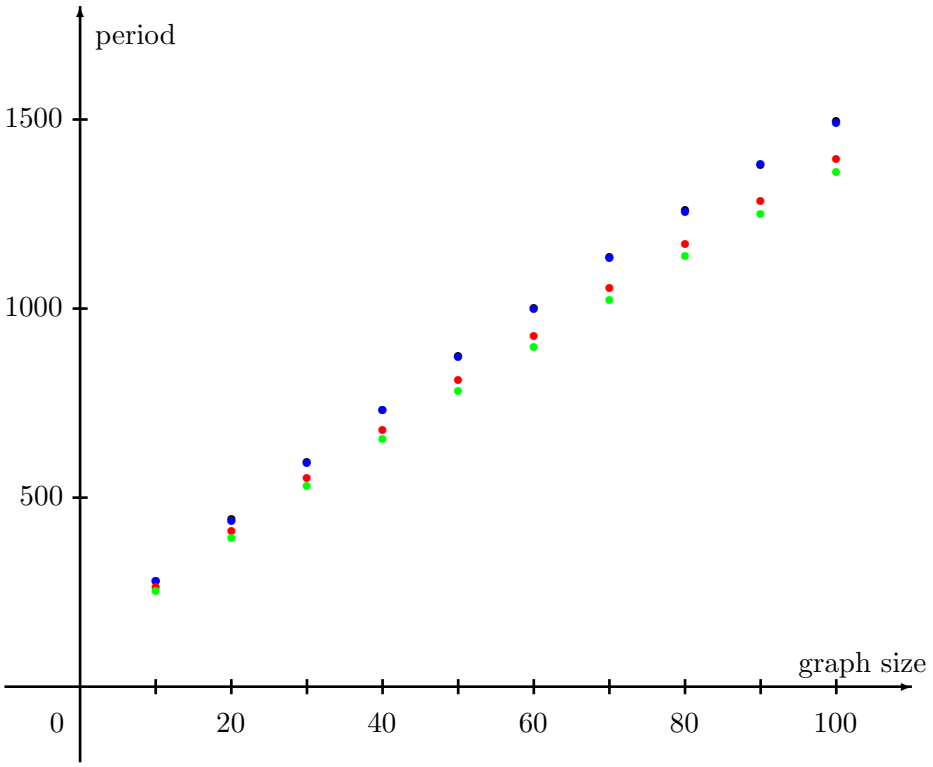
Size of linear graph	70	80	90	100
Number of tests	700	650	600	550
Greedy algorithm	208.027	238.829	266.848	298.616
Communications first algorithm	207.816	238.658	266.713	298.533
Longest first algorithm	207.961	238.658	266.74	298.576
Cycle-time	162.231	182.638	202.987	225.824



Some average results, for random tests, when the platform is fully-homogeneous and consists of $p = 10$ processors, with same speed $s = 1$ and same bandwidth $b = 1$. Card limitations are $B^o = B^i = 1$. We took $(a, b) = (0, 99)$ and $(c, d) = (0, 0)$, which means that all computation's size are 0.

Size of linear graph	10	20	30	40	50	60
Number of tests	1000	950	900	850	800	750
Greedy algorithm	280.28	442.379	594.581	732.531	874.959	1000.54
Communications first algorithm	278.35	439.327	590.979	731.294	871.236	998.765
Longest first algorithm	263.19	412.415	552	678.866	810.486	926.949
Cycle-time	253.028	393.693	530.894	655.529	782.05	898.947

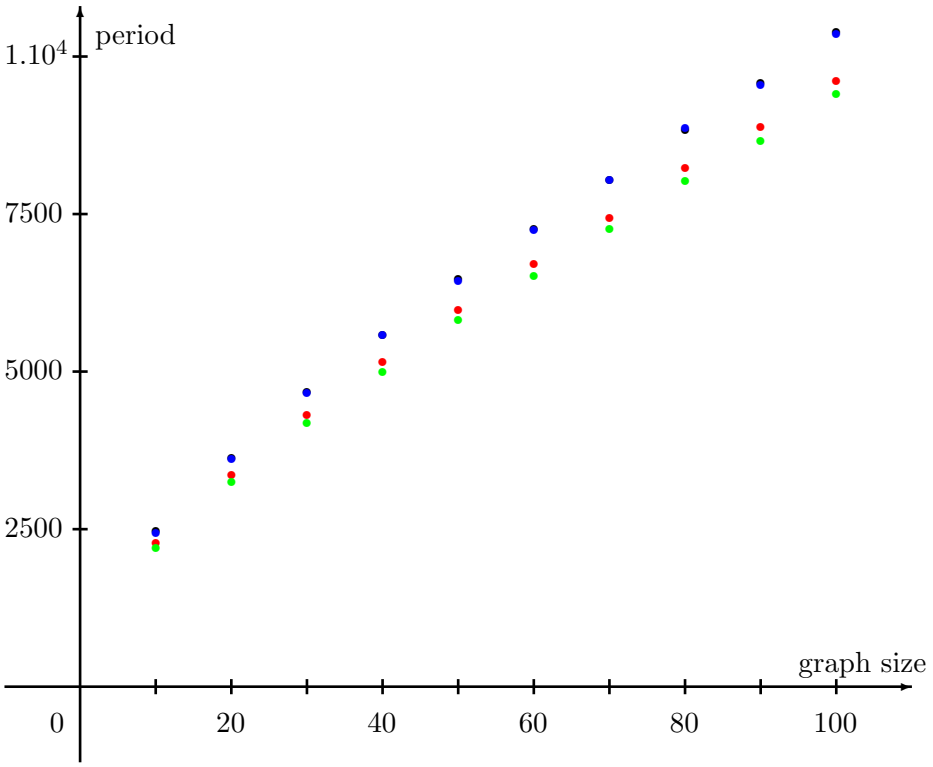
Size of linear graph	70	80	90	100
Number of tests	700	650	600	550
Greedy algorithm	1135.43	1259.89	1380.43	1495.6
Communications first algorithm	1132.85	1255.84	1382.31	1491.55
Longest first algorithm	1053.61	1169.55	1285.32	1394.94
Cycle-time	1023.43	1139.5	1250.77	1361.12



Some average results, for random tests, when the platform is fully-homogeneous and consists of $p = 20$ processors, with same speed $s = 1$ and same bandwidth $b = 1$. Card limitations are $B^o = B^i = 1$. We took $(a, b) = (0, 999)$ and $(c, d) = (0, 0)$, which means that all computation's size are 0.

Size of linear graph	10	20	30	40	50	60
Number of tests	1000	950	900	850	800	750
Greedy algorithm	2468.42	3630.3	4673.57	5584.81	6462.88	7265.42
Communications first algorithm	2440.9	3612.76	4661.92	5574.18	6430.19	7253.8
Longest first algorithm	2278.46	3364.7	4318.16	5147.7	5982.88	6700.21
Cycle-time	2196.36	3240.2	4186.05	4995.31	5816.46	6519.81

Size of linear graph	70	80	90	100
Number of tests	700	650	600	550
Greedy algorithm	8034.79	8841.3	9577.88	10388.3
Communications first algorithm	8038.57	8868.45	9540.42	10360.2
Longest first algorithm	7436.78	8226.86	8875.82	9617.78
Cycle-time	7268.89	8027.19	8665.19	9400.26



In the next sections, we use some examples where tasks have exponential sizes. Because of this, we test previous algorithms on random graphs where the number of small tasks is exponentially bigger than the number of big tasks, and where big sizes are exponentially bigger than small ones. To create such graphs, we use a gaussian law for choosing the size of the tasks.

A good way to generate a gaussian law is to use the Box-Muller method: if x_1 and x_2 are independent and uniformly picked in $]0,1[$, $y_1 = \sqrt{-2\ln(x_1)}\sin(2\pi x_2)$ and $y_2 = \sqrt{-2\ln(x_1)}\cos(2\pi x_2)$ follow a standard normal distribution $\mathcal{N}(0,1)$, and $z_1 = m + sy_1$ and $z_2 = m + sy_2$ follow the normal distribution $\mathcal{N}(m,s)$.

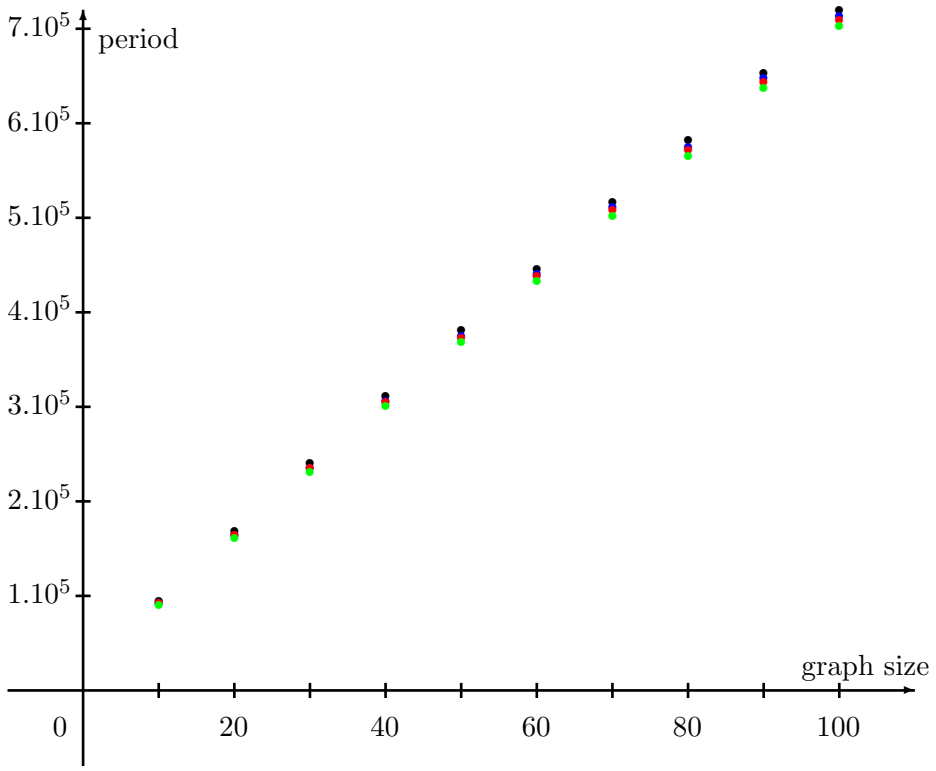
We don't want negative sizes, so we construct $y_1' = \sqrt{-2\ln(x_1)}\sin(\pi x_2)$ and $z_1' = m + sy_1'$. This law is exactly the law $\mathcal{N}(0,1)$ conditioned to be positive, and is also the law $|\mathcal{N}(0,1)|$.

We have to choose the values m and s. In some future examples, the size of tasks increase as their number decrease. Because of this, we can choose $m = 0$. Moreover, if X follows a law $\mathcal{N}(0,s)$, we know that $P(0 \leqslant X \leqslant s | X \leqslant 0) \approx 0.68$, $P(0 \leqslant X \leqslant 2s | X \leqslant 0) \approx 0.95$ and $P(0 \leqslant X \leqslant 3s | X \leqslant 0) \approx 0.997$. If we choose $3s = 3000$, we have in average about 5 tasks, for a graph of size 100 whose sizes are between 2000 and 3000, which is quite similar to the next examples. As a consequence, we will use the law $|\mathcal{N}(0,1000)|$ to generate random sizes on our graphs.

Some average results, for random tests, when the platform is fully-homogeneous and consists of $p = 3$ processors, with same speed $s = 1$ and same bandwidth $b = 1$. Card limitations are $B^o = B^i = 1$.

Size of linear graph	10	20	30	40	50	60
Number of tests	1000	950	900	850	800	750
Greedy algorithm	9508.23	16886.6	24092.4	31151.8	38079.7	44565.9
Communications first algorithm	9198.17	16451.1	23583.1	30632.1	37493.8	44031.3
Longest first algorithm	9197.92	16410	23483.2	30495.7	37326.8	43792.5
Cycle-time	9016.71	16120.7	23108	30061.5	36814.7	43272.9

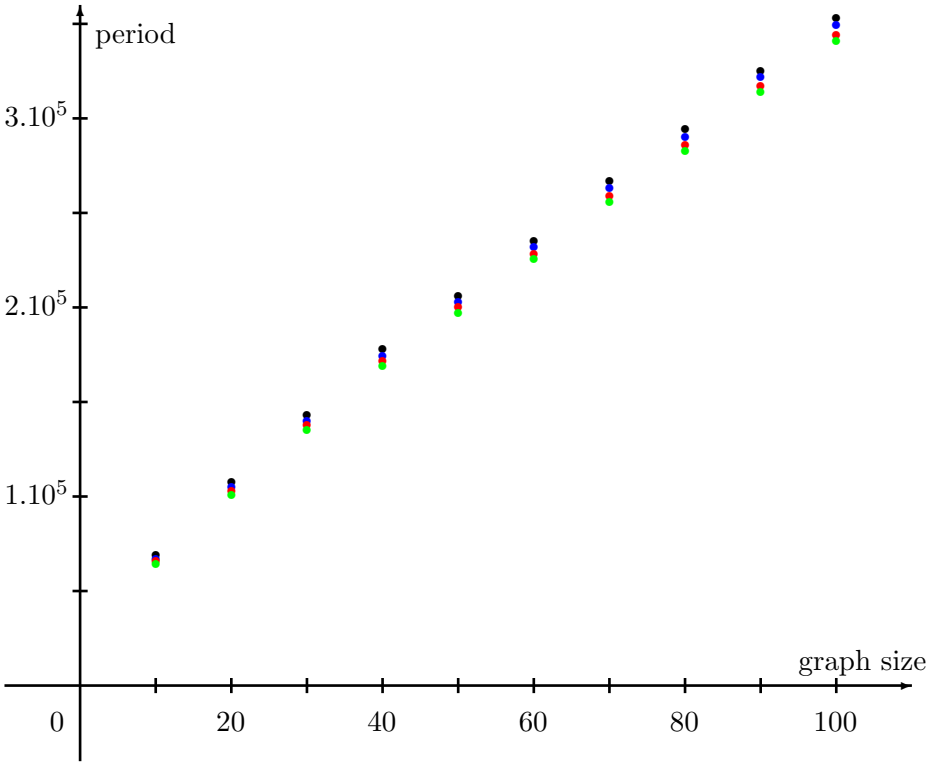
Size of linear graph	70	80	90	100
Number of tests	700	650	600	550
Greedy algorithm	51706.2	58239.3	65367.3	71981.3
Communications first algorithm	51099.6	57459.1	64771.1	71317.1
Longest first algorithm	50787.9	57130.6	64386.7	70918.1
Cycle-time	50233.9	56562.2	63759	70277.5



Some average results, for random tests, when the platform is fully-homogeneous and consists of $p = 10$ processors, with same speed $s = 1$ and same bandwidth $b = 1$. Card limitations are $B^o = B^i = 1$.

Size of linear graph	10	20	30	40	50	60
Number of tests	1000	950	900	850	800	750
Greedy algorithm	6919.04	10771.3	14335	17789.5	20629.7	23527.1
Communications first algorithm	6680.3	10481.7	14006.9	17455	20313.3	23182.6
Longest first algorithm	6597.3	10305.3	13769.3	17162.4	20003.8	22841.7
Cycle-time	6432.42	10095.3	13542.7	16925.2	19728.3	22538.4

Size of linear graph	70	80	90	100
Number of tests	700	650	600	550
Greedy algorithm	26691.9	29412.7	32525.5	35320.3
Communications first algorithm	26294.8	29029	32179.5	34919.2
Longest first algorithm	25885.6	28584.1	31692.1	34396.3
Cycle-time	25576.5	28267.5	31382.7	34067.2



For these tests using random graphs, it seems that the Greedy algorithm, the Computations First algorithm and the Longest First algorithm compute schedules with small periods. More exactly, generally, the period computed by these algorithms is only slightly larger than the cycle-time, which is a lower bound on the period. Because of this, it makes sense to investigate whether these algorithms provide a k-approximation (for some constant k) for the period, and whether there are links between the optimal period $K_{\min}$ and the cycle-time CT.

4.2.6 Greedy algorithm and the Communications First algorithm do not provide a k-approximation (for any constant k) of the cycle-time

In spite of the first intuition given by previous experiments, we now show that the Greedy algorithm and the Communications First algorithm do not always compute schedules of periods similar to cycle-times.

Proposition 4.3. *For any constant k, there exists a linear graph G and a mapping a such that, if K_1 (resp. K_2) is the period of the schedule computed by the Greedy algorithm (resp. the Communications First algorithm), we have:*

$$K_1 \geqslant k.CT(a, G)$$

$$K_2 \geqslant k.CT(a, G)$$

where $CT(a, G)$ is the cycle-time of the linear graph G and the mapping a.

Remark. Since the cycle-time time is only a lower bound of the optimal period, this do not prove that the Greedy algorithm and the Communications First algorithm are not some k-approximation for the period.

Proof. To prove the previous theorem, we provide a construction of some counter examples.

We assume that the platform is fully-homogeneous and consists of $p = n$ processors $P_1, P_2, \ldots, P_n$ of same speeds $s = 1$. Network cards capacities are $B^i = B^o = 1$ and links between processors have same bandwidths $b = 1$.

Our counter examples only contain communications between processors (computations are of size 0 and we will omit them). This way, these counter examples fit either the Greedy algorithm than the Communications First one. We also omit communications of size 0, which means a communication $com(P_k, P_l, s_2)$ can follow a communication $com(P_i, P_j, s_1)$ even if $P_j \neq P_k$ (in fact, between these communications we have a communication $com(P_j, P_k, 0)$ in the corresponding linear graph).

Let us suppose that we have $(n-1)$ strictly positive integers $\{k_1, k_2, \ldots k_{n-1}\}$ such that,

$$\forall i \in \{1, \ldots, n-2\}, k_{i+1} < k_i \frac{n-i}{(n-i-1)^2} \tag{11}$$

We can construct these integers from k_{n-1} to k_1 by induction, using the formula

$$k_i = \lfloor k_{i+1} \frac{(n-i-1)^2}{n-i} \rfloor + 1$$

and starting with $k_{n-1} = 1$. One can check that k_1 is at least exponential in n.

Using this formula, for $n = 15$, we obtain:

$\{k_1, \ldots, k_{14}\} =$
$\{1582973825, 131133929, 11838479, 1174064, 129147, 15944, 2242, 366, 71, 17, 5, 2, 1, 1\}$

Let us now describe the list of communications of the counter example step by step. Firstly, there is k_1 times the following set of communications:

$$\left\{ com\left(P_1, P_n, \frac{1}{k_1(n-1)}\right), com\left(P_1, P_{n-1}, \frac{1}{k_1(n-1)}\right), \ldots, com\left(P_1, P_2, \frac{1}{k_1(n-1)}\right) \right\}$$

The Greedy algorithm or the Communications First one construct for these $k_1(n-1)$ tasks a schedule represented in Figure 21, where any processor P_i for $i \in \{2, \ldots, n\}$ has k_1 communications of size $\frac{1}{k_1}\frac{1}{n-1}$ separated by gaps of size $\frac{1}{k_1}\frac{n-2}{n-1}$ in the schedule.

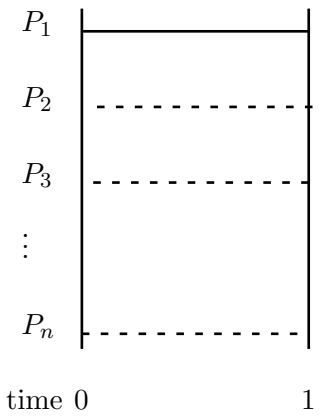


Figure 21: Construction of a schedule in order to prove Proposition 4.3.

Then, there is in the counter example k_2 times the following set of communications:

$$\left\{ com\left(P_2, P_n, \frac{1}{k_2(n-2)}\right), com\left(P_2, P_{n-1}, \frac{1}{k_2(n-2)}\right), \ldots, com\left(P_2, P_3, \frac{1}{k_2(n-2)}\right)\right\}$$

We supposed that $k_2 < k_1\frac{n-1}{(n-2)^2}$, which means that $\frac{1}{k_2(n-2)} > \frac{1}{k_1}\frac{n-2}{n-1}$. Gaps of size $\frac{1}{k_1}\frac{n-2}{n-1}$ on the previous schedule are too small to put any task of size $\frac{1}{k_2(n-2)}$. Because of this, the schedule computed by the Greedy algorithm (or by the Communications First algorithm) looks like Figure 22. In the schedule, any processor P_i, for $i \in \{3, \ldots, n\}$, has k_2 communications of size

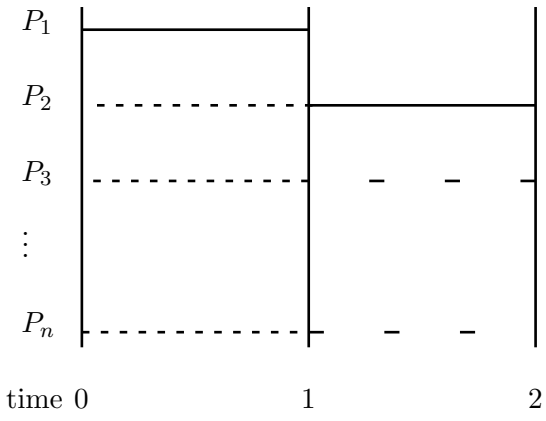


Figure 22: Construction of a schedule in order to prove Proposition 4.3.

$\frac{1}{k_2}\frac{1}{n-2}$ separated by gaps of size $\frac{1}{k_2}\frac{n-3}{n-2}$ for $1 \leqslant \text{time} \leqslant 2$.

By induction, this construction is repeated for $\{k_3, k_4, \ldots, k_{n-1}\}$. At the end the schedule computed by the Greedy algorithm (resp. the Communications First algorithm) looks as Figure 23. For this example, the Greedy algorithm (resp. the Communications First algorithm)

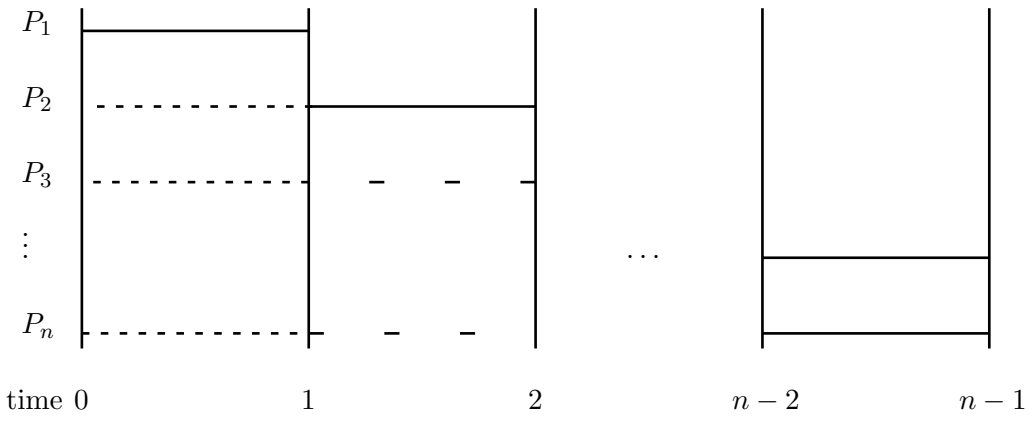


Figure 23: Construction of a schedule in order to prove Proposition 4.3.

computes a schedule of period $(n-1) = O(n)$. Moreover, the cycle-time CT follows:

$$CT = \sum_{i=1}^{n-1} k_i \frac{1}{k_i(n-i)} = \sum_{i=1}^{n-1} \frac{1}{(n-i)} = \sum_{j=1}^{n-1} \frac{1}{j} = O(\log(n))$$

This shows that there exist examples where the schedules found by the Greedy algorithm (resp. the Communications First algorithm) have not a period smaller than k times the cycle-time for any constant k. □

We can run previous algorithms on the example detailed in this part. The size of our example increase very quickly with the number of processors n. Because of this, we give some results of periods found by previous algorithms when $2 \leqslant n \leqslant 7$.

Number of processors n	2	3	4	5	6	7	n
Size of the graph	1	5	17	57	227	1079	$O((n+1)!)$
Greedy algorithm	1	2	3	4	5	6	$n-1$
Communications first algorithm	1	2	3	4	5	6	$n-1$
Longest first algorithm	1	2	2	2.18	2.41	2,5	??? (Given later)
Cycle-time	1	1.5	1.83	2.08	2.28	2.45	$O(\log(n))$

In the following part, we construct an other example, which prove that the greedy algorithm and the first modified one are not some k-approximations, for a constant k.

4.2.7 Greedy algorithm and the Communications First algorithm do not provide a k-approximation (for any constant k) of the optimal period

Proposition 4.3 gives a relation between cycle-time and periods founds by the Greedy algorithm and the Communications First one. This section extends this to the optimal period.

Proposition 4.4. *There is no constant k such that the Greedy algorithm (resp. the Communications First algorithm) computes a schedule whose period is a k-approximation of the optimal period.*

Proof. As in Proposition 4.3's proof, we give a linear graph and a mapping, then compute corresponding schedules using the Greedy or the Communications First algorithm and extract periods of these schedules. The main difference is that instead of making comparisons between

these periods and the cycle-time, we give an other schedule that has a small period, and compare periods to periods. Counter examples using for this proof are similar to those used perviously.

Construction of the counter example

We assume that our platform is fully-homogeneous and consists of $2n$ processors $P_1, P_2, \ldots, P_{2n}$ of same speeds $s = 1$. Network cards capacities are $B^i = B^o = 1$ and links between processors have same bandwidths $b = 1$.

As in the previous part, our example only contains communications between processors (computations are of size 0 and we will omit them). We also omit to write communications of size 0.

Let ε be a constant in $]0, 1/2[$. Let us suppose that we have $(n-1)$ strictly positive integers $\{k_1, k_2, \ldots k_{n-1}\}$ such that,

$$\forall i \in \{1, \ldots, n-2\}, \quad k_i > \frac{1}{\varepsilon} \frac{(n-i-\varepsilon)(n-i-1)}{n-i} k_{i+1} \tag{12}$$

We can construct these integers from k_{n-1} to k_1 by induction, using the formula

$$k_i = \lfloor \frac{1}{\varepsilon} \frac{(n-i-\varepsilon)(n-i-1)}{n-i} k_{i+1} \rfloor + 1$$

and starting with $k_{n-1} = 1$. Ones can check that k_1 is at least exponential in n and in $\frac{1}{\varepsilon}$.

We prose

$$\forall i \in \{1, 2, \ldots, n-1\}, \quad \varepsilon_i = \frac{\varepsilon}{(n-i)k_i}, \quad \delta_i = \frac{1-\varepsilon}{k_i} \tag{13}$$

Since this proof is quite similar to Proposition 4.3's one, we only describe the induction part. Let us suppose that, at a stage $i \in \{2, \ldots, n-2\}$ the schedule looks like Figure 24. We

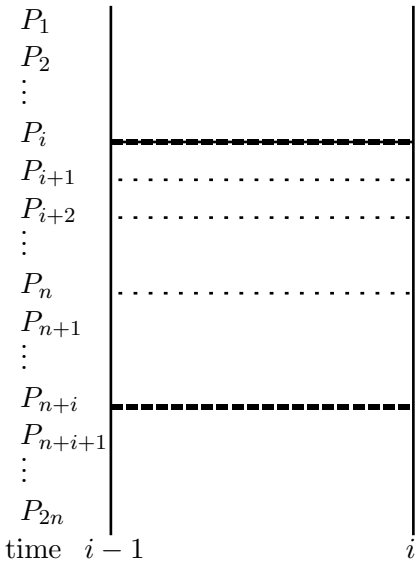


Figure 24: Construction of a schedule in order to prove Proposition 4.4.

assume that, in the schedule constructed by the Greedy algorithm (resp. the Communications First algorithm), in the interval of time $[t-1,t[, t \leqslant i$, a processor $P_j, j \in \{t+1, t+2, \ldots, n\}$ has k_t communications with P_t of equal size ε_t separated by gaps of same size $\frac{n-t-\varepsilon}{(n-t)k_t}$ (in blue for $t = i$). Ones can check that the sum of tasks and gaps sizes is equal to 1:

$$k_t\varepsilon_t + k_t\frac{n-t-\varepsilon}{(n-t)k_t} = \frac{k_t\varepsilon}{k_t(n-t)} + \frac{k_t(n-t-\varepsilon)}{k_t(n-t)} = 1$$

Processor P_{n+t} has k_t communications with P_t of equal size δ_t separated by gaps of same size $\frac{1}{k_t} - \delta_t$ (in red for $t = i$). Once again, the sum of these sizes is equal to 1:

$$k_t\delta_t + k_t\left(\frac{1}{k_t} - \delta_t\right) = 1$$

Using the greedy algorithm, we add to this schedule k_{i+1} times the following list of communications:

$$\{\mathbf{com}\,(\boldsymbol{P_{i+1}, P_{n+i+1}, \delta_{i+1}}), com\,(P_{i+1}, P_{i+2}, \varepsilon_{i+1})\,, com\,(P_{i+1}, P_{i+3}, \varepsilon_{i+1})\,, \ldots, com\,(P_{i+1}, P_n, \varepsilon_{i+1})\}$$

We want to prove that the Greedy algorithm (resp. the Communications First algorithm) put these communications in the interval of time $[i, i+1[$. This is true if for any communication $com\,(P_a, P_b, s)$, s is bigger than gaps on processor P_a in the schedule in the interval of time $[0, i[$. We now show that we are in that case.

The size of a communication $\mathbf{com}\,(\boldsymbol{P_{i+1}, P_{n+i+1}, \delta_{i+1}})$ is δ_{i+1}. In the schedule and for processor P_{i+1}, gaps on the interval of time $[t, t+1[, t \in \{1, \ldots, i-1\}$ are of size $\frac{n-t-\varepsilon}{(n-t)k_t}$. Moreover, we have:

$$k_i > \frac{1}{\varepsilon}\frac{(n-i-\varepsilon)(n-i-1)}{n-i}k_{i+1} \quad (A) \qquad \text{by (12)}$$

$$(A) \implies k_i > \frac{1}{1-\varepsilon}\frac{n-i-\varepsilon}{n-i}k_{i+1} \quad (B) \qquad \text{because } \varepsilon \in [0, 1/2[\text{ and } (n-i-1) \geqslant 1$$

$$(B) \iff \frac{1-\varepsilon}{k_{i+1}} > \frac{n-i-\varepsilon}{(n-i)k_i} \quad (C)$$

$$(C) \implies \frac{1-\varepsilon}{k_{t+1}} > \frac{n-t-\varepsilon}{(n-t)k_t} \quad (D) \qquad \text{because } i \geqslant t \text{ and } k_t \geqslant k_i$$

$$(D) \iff \delta_{i+1} > \frac{n-t-\varepsilon}{(n-t)k_t} \qquad \text{by (13)}$$

which prove that gaps are smaller that the communication we want to add.

Likewise, the size of any task $com\,(P_{i+1}, P_k, \varepsilon_{i+1})\,, k \in \{i+2, \ldots, n\}$ is ε_{i+1}, and we have:

$$k_i > \frac{1}{\varepsilon}\frac{(n-i-\varepsilon)(n-i-1)}{n-i}k_{i+1} \quad (A') \qquad \text{by (12)}$$

$$(A') \implies \frac{\varepsilon}{(n-i-1)k_{i+1}} > \frac{n-t-\varepsilon}{(n-t)k_t} \quad (B') \qquad \text{because } i \geqslant t \text{ and } k_t \geqslant k_i$$

$$(B') \iff \varepsilon_{i+1} > \frac{n-t-\varepsilon}{(n-t)k_t} \qquad \text{by (13)}$$

which prove that, once again, gaps are smaller that the communication we want to add.

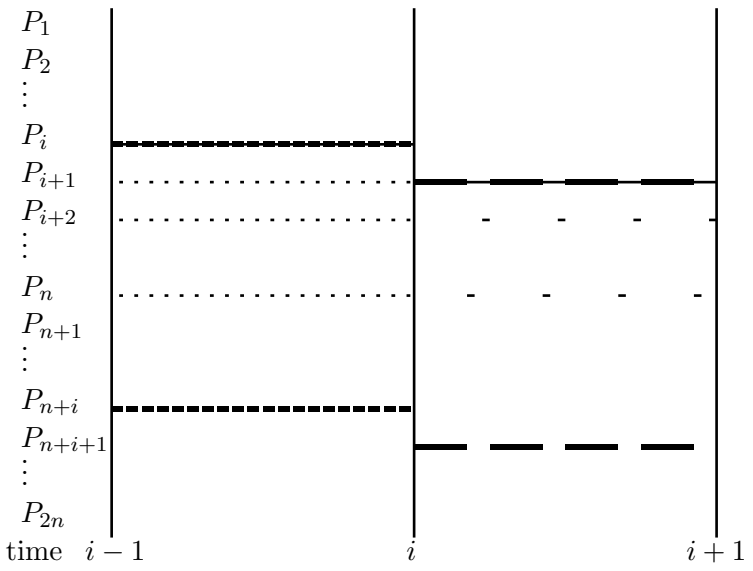


Figure 25: Construction of a schedule in order to prove Proposition 4.4.

With these $k_{i+1}(n-i)$ new communications, the schedule looks like Figure 25. When the induction ends, the schedule computed by the Greedy algorithm (resp. the Communication First algorithm) has a period of $n-1$. In order to complete the proof, we have to give an other schedule which has a period $o(n)$.

An other schedule for the previous list of communications

The idea is very simple. We first add in the schedule communications $\mathbf{com}\,(\boldsymbol{P_i}, \boldsymbol{P_{n+i}}, \boldsymbol{\delta_i})$ for $i \in \{1, \ldots, n-1\}$. Two communications $\mathbf{com}\,(\boldsymbol{P_{i_1}}, \boldsymbol{P_{n+i_1}}, \boldsymbol{\delta_{i_1}})$ and $\mathbf{com}\,(\boldsymbol{P_{i_2}}, \boldsymbol{P_{n+i_2}}, \boldsymbol{\delta_{i_2}})$ don't share a processor if $i_1 \neq i_2$. Because of this, two communications $\mathbf{com}\,(\boldsymbol{P_{i_1}}, \boldsymbol{P_{n+i_1}}, \boldsymbol{\delta_{i_1}})$ and $\mathbf{com}\,(\boldsymbol{P_{i_2}}, \boldsymbol{P_{n+i_2}}, \boldsymbol{\delta_{i_2}})$ can be computed in parallel in the schedule if $i_1 \neq i_2$.

Moreover, there is k_i communications $\mathbf{com}\,(\boldsymbol{P_i}, \boldsymbol{P_{n+i}}, \boldsymbol{\delta_i})$. We add them one per one in the schedule, it takes a size of $k_i\delta_i$ and we have:

$$k_i\delta_i = k_i\frac{1-\varepsilon}{k_i} = 1-\varepsilon < 1$$

The schedule containing all communications $\mathbf{com}\,(\boldsymbol{P_i}, \boldsymbol{P_{n+i}}, \boldsymbol{\delta_i})$ for $i \in \{1, \ldots, n-1\}$ has a size smaller than 1. We also have to add in the schedule communications $com\,(P_i, P_j, \varepsilon_j)$, $i \in \{1, \ldots, n-1\}$ and $j \in \{i+1, \ldots, n\}$. The sum of these communications sizes is $\sum_{i=1}^{n-1} k_i(n-i-1)\varepsilon_i$ and we have:

$$\sum_{i=1}^{n-1} k_i(n-i-1)\varepsilon_i = \sum_{i=1}^{n-1} k_i(n-i-1)\frac{\varepsilon}{(n-i)k_i} = \varepsilon\sum_{i=1}^{n-1}\frac{n-i-1}{n-i} < \varepsilon(n-1)$$

We can choose ε in $]0, 1/2[$. Letting $\varepsilon = \min(\frac{1}{n-1}, 1/2)$, this sum is smaller than 1. We can add all these communications in the schedule one per one in the interval of time $[1, 2[$. This method constructs a schedule as a period smaller than 2.

Finally, for any $n \in \mathbb{N}$, there exists a linear graph and a mapping such that the Greedy algorithm (resp. the Communications First algorithm) computes a schedule of period $(n-1)$, and whose minimal period $K_{\min}$ is smaller than 2. This ends the proof. □

Remark. Letting n be 15 (i.e. on $p = 30$ processors), we have $k_1 \sim 10^{25}$. Counter examples used in the previous proof exists, but are not very common in "real life".

4.2.8 Periods found by the Longest First algorithm are some 4-approximation of the cycle-time

Proposition 4.4 states that the Greedy algorithm and the Communications First one are not k-approximations of the optimal period. These results are interesting because not intuitive, but they do not give any method to compute schedules with small periods. The end of this section is devoted to prove that the Longest First algorithm constructs a schedule with "small" periods.

Lemma 4.5. *The Longest First algorithm computes schedules whose periods are 4-approximation of cycle-times.*

Proof. Let us recall that the Longest First algorithm (see Algorithm 3) adds tasks, communications and computations, *from the longest to the shortest* as soon as possible in the schedule. Because of this, in this proof, we suppose that these tasks are given by the ordered list $(t_i)_{1 \leqslant i \leqslant N}$, t_1 is the longest task and t_N the shortest one. We assume that the size of task t_i is given by $\text{size}(t_i)$.

We introduce some other notations. We assume that the schedule contains the first k tasks $t_1, \ldots, t_k$. For any processor P_i, $1 \leqslant i \leqslant p$, the ending work's time is $E_k(i)$. We know that the schedule is of period K for any $K \geqslant E_k$, with $E_k = \max_{1 \leqslant i \leqslant p} \{E_k(i)\}$. These notations are represented on Figure 26.

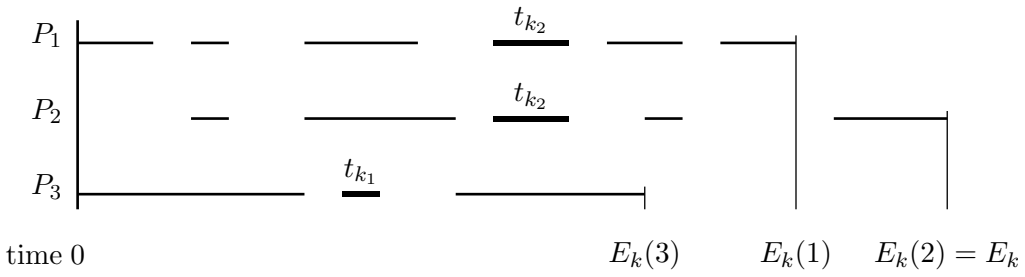


Figure 26: A simple schedule for representing new notations when the platform consists of 3 processors P_1, P_2 and P_3. We suppose that there are k tasks in this schedule. Two tasks are represented in bold: a **computation** t_{k_1} and a **communication** t_{k_2}.

We now prove by induction that, $\forall\ 0 \leqslant k \leqslant N$,

$$E_k \leqslant 4.CT \tag{14}$$

Initialisation: Equation 14 is clearly verified when $k = 0$ because there is no task in the schedule., and $E_0 = 0 \leqslant 4.CT$.

Induction part: Let us suppose that the algorithm has constructed, with the first k tasks, a schedule such that equation 14 is verified. When the Longest First algorithm adds the task

t_{k+1} of size size(t_{k+1}), there are two possibilities: it can modify E_k or not.

Let us first suppose that E_k is not modified, i.e. $E_{k+1} = E_k$. By hypothesis, $E_k \leqslant 4.CT$, i.e. $E_{k+1} \leqslant 4.CT$, and the induction works.

Let us now suppose that $E_{k+1} \neq E_k$. Clearly, $E_{k+1} > E_k$, because adding a task t_{k+1} can only increase the ending work's time of a processor.

$$\begin{aligned} E_{k+1} > E_k &\iff \max_{1 \leqslant l \leqslant p} \{E_{k+1}(l)\} > \max_{1 \leqslant l \leqslant p} \{E_k(l)\} \\ &\iff \exists\ i_1, i_2 \in \{1, \ldots, p\}^2, \begin{cases} E_{k+1}(i_2) > E_k(i_1) \geqslant E_k(i_2) \\ \max_{1 \leqslant l \leqslant p}\{E_k(l)\} = E_k(i_1) \\ \max_{1 \leqslant l \leqslant p}\{E_{k+1}(l)\} = E_{k+1}(i_2) \end{cases} \end{aligned}$$

These equations mean that at stage k, these is a processor P_{i_1} such that $E_k = E_k(i_1)$. When we add the new task t_{k+1} on the processor P_{i_2}, the algorithm put this task at the end of P_{i_2}'s schedule, and increase the ending time of processor P_{i_2} from $E_k(i_2)$ to $E_{k+1}(i_2) \geqslant E_k(i_2) + \text{size}(t_{k+1})$. (This is not an equality because it's possible that the algorithm leave a gap on processor P_{i_2} before putting the task t_{k+1}.)

Let us suppose that the task t_{k+1} is a communication between processor P_{i_2} and a different processor P_{i_3}. The schedule at stage $k+1$ is represented in Figure 27.

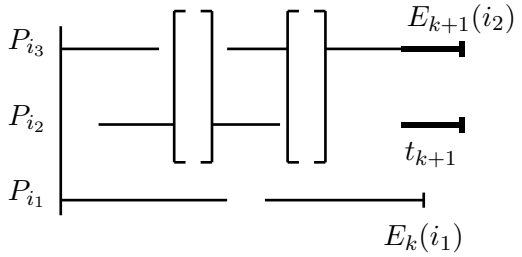


Figure 27: Addition of a **communication** in a schedule using the Longest First algorithm. Common gaps between processors P_{i_2} and P_{i_3} are represented between brackets.

The number of common gaps $\#_{\text{common gaps}}(P_{i_2}, P_{i_3}, k+1)$ between processors P_{i_2} and P_{i_3} after adding the task t_{k+1} in the schedule (represented between brackets in Figure 27) in bounded by the number of tasks on P_{i_2} plus the number of tasks on P_{i_3} at stage $k+1$.

$$\#_{\text{common gaps}}(P_{i_2}, P_{i_3}, k+1) \leqslant \#(l \leqslant k+1, t_l \text{ is on } P_{i_2}) + \#(l \leqslant k+1, t_l \text{ is on } P_{i_3}) \qquad (15)$$

Moreover, there is no common gap in the schedule of size bigger than size(t_{k+1}), because the Longest First algorithm would have put the task t_{k+1} in it. The size of the sum of common gaps $S(P_{i_2}, P_{i_3}, k+1)$ between P_{i_2} and P_{i_3} at stage $k+1$ in bounded by the number of common gaps times the biggest size of a common gap:

$$S(P_{i_2}, P_{i_3}, k+1) \leqslant \text{size}(t_{k+1}).\#_{\text{common gaps}}(P_{i_2}, P_{i_3}, k+1) \qquad (16)$$

Combining (15) and (16) follows

$$S(P_{i_2}, P_{i_3}, k+1) \leqslant \text{size}(t_{k+1}).\#\,(l \leqslant k+1, t_l \text{ is on } P_{i_2}) + \text{size}(t_{k+1}).\#\,(l \leqslant k+1, t_l \text{ is on } P_{i_3}) \qquad (17)$$

For all $l \leqslant k, \text{size}(t_{k+1}) \leqslant \text{size}(t_l)$. Because of this, we have:

$$\text{size}(t_{k+1}).\#\left(l \leqslant k, t_l \text{ is on } P_{i_2}\right) \leqslant \sum_{l=1}^{k+1} \text{size}(t_l)\mathbb{1}_{t_l \text{ is on } P_{i_2}} \leqslant CT_{i_2}$$

$$\text{size}(t_{k+1}).\#\left(l \leqslant k, t_l \text{ is on } P_{i_3}\right) \leqslant \sum_{l=1}^{k+1} \text{size}(t_l)\mathbb{1}_{t_l \text{ is on } P_{i_3}} \leqslant CT_{i_3}$$

Combining these equations with (17) follows

$$S(P_{i_2}, P_{i_3}, k+1) \leqslant CT_{i_2} + CT_{i_3} \leqslant 2 \max_{1 \leqslant i \leqslant p} \{CT_i\} = 2.CT \tag{18}$$

The value of $E_{k+1}(i_2)$ (see Figure 27) is smaller than the size of the common gaps between P_{i_2} and P_{i_3} at stage $k+1$ (represented in green) plus the sum of all the tasks on P_{i_2} plus the sum of all the tasks on P_{i_3}. This can be written

$$E_{k+1}(i_2) \leqslant S(P_{i_2}, P_{i_3}, k+1) + CT_{i_2} + CT_{i_3} \tag{19}$$

Combining this equation with (18) follows

$$E_{k+1} = E_{k+1}(i_2) \leqslant 2.CT + CT_{i_2} + CT_{i_3} \leqslant 4.CT \tag{20}$$

which ends the induction when the task t_{k+1} is a communication between two processors.

When t_{k+1} is a computation on processor P_{i_2}, we can consider that t_{k+1} is a communication between P_{i_2} and a virtual processor P_{i_3} that has an empty schedule, and apply the previous proof. This ends the induction when t_{k+1} is a computation.

Letting $k = N$ in equation 14, we obtain

$$E_N \leqslant 4.CT$$

and the minimal period of this schedule follows $K \leqslant CT \leqslant E_N \leqslant 4.CT$ which ends the proof. □

4.2.9 The Longest First algorithm is a 4-approximation for the period

Lemma 4.5 bounds periods found by the Longest First algorithm. Using this theorem, it is easy to verify that the Longest First algorithm is a 4-approximation for the period.

Theorem 4.6. *The Longest First algorithm computes a schedule whose period is a 4-approximation of the optimal one.*

Proof. With Lemma 4.5 we have $K \leqslant 4.CT$, where K is the period found for a linear graph and a mapping by the Longest First algorithm and CT the cycle-time. Since the cycle-time is smaller than the optimal period $K_{\min}$, we also have $K \leqslant 4.CT \leqslant 4.K_{\min}$, which ends the proof. □

Remark. The Longest First algorithm finds a period that is smaller that $4.CT$. We point at that this result is interesting for someone searching for a mapping and a schedule : with a balanced mapping where the maximum amount of work of a processor is minimized, i.e. where the cycle-time is as small as possible, the Longest First algorithm computes a schedule with a small period. Minimizing the cycle-time when searching for the mapping give some bounds for the period K found by the Longest First algorithm and for the minimal period $K_{\min}$.

4.2.10 The Longest First algorithm is not better than a 4-approximation

Theorem 4.6 states that the Longest First algorithm constructs a schedule whose period is smaller than four times the optimal one. We now prove that unfortunately this algorithm is not a better approximation for the period.

Theorem 4.7. *For any $\beta < 4$, there exists a linear graph and a mapping such that the Longest First algorithm computes a schedule of period K, and $K > \beta.K_{\min}$. In other words, the Longest First algorithm is not a k-approximation for any k smaller than 4.*

Proof. Let us choose $\beta \in [1, 4[$. We explain how ones can construct a linear graph such that the Longest First algorithm constructs a schedule of period K bigger than $\beta K_{\min}$. This proves that this algorithm is not a k-approximation for $k < 4$.

In all this part, we say we have a platform of p processors to say that our platform is fully-homogeneous and consists of p processors. All bandwidths and all speeds are equal to 1.

Some notations

Let N be in $\mathbb{N}$ such that $N \geqslant 2$. We call $H_{2n}(N)$ the fact that there exists a linear graph $G_{2n}(N)$ and a mapping on $p = \alpha_{2n}(N)$ processors[6] such that the schedule computed by the last algorithm looks like Figure 28.

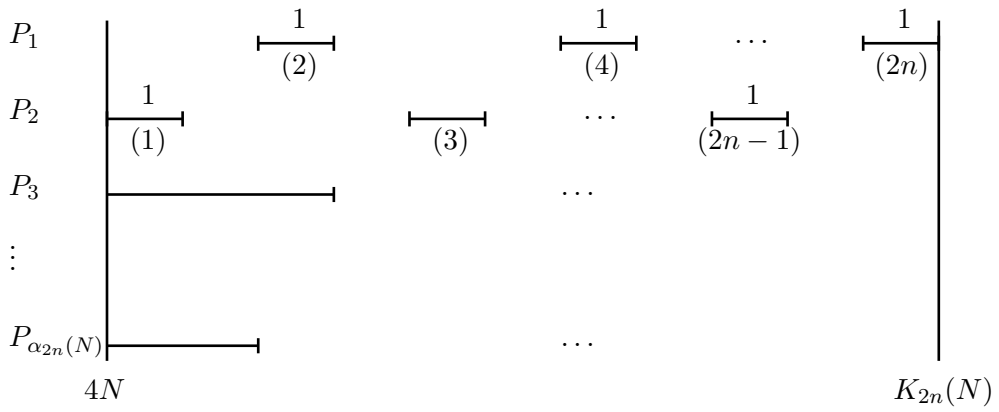


Figure 28: Schedule of property $H_{2n}(N)$.

We suppose in $H_{2n}(N)$ that:

1. Between $t = 0$ and $t = 4N$, there is a computation of size $4N$ on each processor (not represented on the figure).
2. For $t \geqslant 4N$, on P_1, there is firstly a gap of size $(2 - 1/N)$, then a communication of size 1, a gap a size $(3 - 2/N)$, a communication of size 1, a gap a size $(3 - 2/N)$ and so on.
3. The schedule is identical on P_2 without the first gap.

[6]The exact value of $\alpha_{2n}(N)$ is not given and has no importance in the proof.

4. The minimal period of the schedule is $K_{2n}(N)$.

5. There exists an other schedule for the graph $G_{2n}(N)$ such that all tasks on P_1 and P_2 are set in the interval of time $[0, 4N + n]$ (this means that there is no gap in the schedule for P_1 and P_2) and such that the period of this schedule in smaller than $8N + n$.

6. A task of non null size has a size bigger or equal to 1.

Remarks:

- Common gaps of P_1 and P_2 are all of size $(1 - 1/N)$.
- $K_{2n}(N) = 4N + 4n - 1 - \frac{2n-1}{N}$

Similarly, we call $H_{2n+1}(N)$ the fact that there exists a linear graph $G_{2n+1}(N)$ and a mapping on $\alpha_{2n+1}(N)$ processors[7] such that the schedule computed by the last algorithm looks like Figure 29.

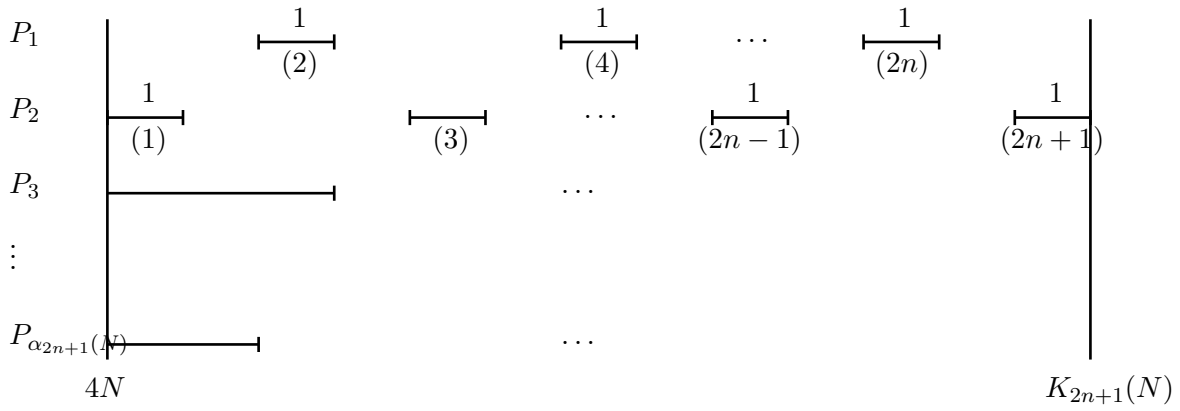


Figure 29: Schedule of property $H_{2n+1}(N)$.

We suppose in $H_{2n+1}(N)$ that:

1. Between $t = 0$ and $t = 4N$, there is a computation of size $4N$ on each processor (not represented on the figure).

2. For $t \geqslant 4N$, on P_1, there is firstly a gap of size $(2 - 1/N)$, then a communication of size 1, a gap a size $(3 - 2/N)$, a communication of size 1, a gap a size $(3 - 2/N)$ and so on.

3. The schedule is identical on P_2 without the first gap and with one more communication.

4. The minimal period of the schedule is $K_{2n+1}(N)$.

5. There exists an other schedule for the graph $G_{2n+1}(N)$ such that all tasks on P_1 are set in the interval of time $[0, 4N + n]$ and all tasks on P_2 are set in the interval of time $[0, 4N + n + 1]$ (this means that there is no gap in the schedule for P_1 and P_2) and such that the period of this schedule in smaller than $8N + n$.

[7] Once again, the exact value of $\alpha_{2n+1}(N)$ is not given and has no importance in the proof.

6. A task of non null size has a size bigger or equal to 1.

Remarks:

- Common gaps of P_1 and P_2 are all of size $(1 - 1/N)$.
- $K_{2n+1}(N) = 4N + 4n + 1 - \frac{2n}{N}$

Now, we show by induction that for all n we have $(H_{2n}(N), H_{2n+1}(N))$. We first explain why this is true for $n \in \{0, 1, \ldots, N\}$, and then we show by strong induction that for $n \geqslant N$, $\{(H_{2k}(N), H_{2k+1}(N))\}_{0 \leqslant k \leqslant n}$ implies $\{(H_{2k}(N), H_{2k+1}(N))\}_{0 \leqslant k \leqslant n+1}$.

Initialization

If $n = 0$, we have $H_0(N)$ with the graph containing only a computation of size $4N$ mapped on P_1. We also have $H_1(N)$ with a graph containing three computations of size $4N$ mapped on P_1, P_2 and P_3, and one communication of size 1 between P_2 and P_3.

More generally, if $n \leqslant N$, we have $H_{2n}(N)$ on $2n + 2$ processors with the graph containing

- (2n+2) computations of size $4N$ respectively mapped on $P_1, \ldots, P_{2n+2}$
- for all $k \in \{1, \ldots, 2n-1\}$, one computation of size $(2 - 1/N)k$ mapped on processor P_{k+3}
- for all $k \in \{1, \ldots, n\}$, one communication of size 1 between P_2 and P_{2k+1}
- for all $k \in \{1, \ldots, n\}$, one communication of size 1 between P_1 and P_{2k+2}

and we have $H_{2n+1}(N)$ on $2n + 3$ processors with a graph containing

- (2n+3) computations of size $4N$ respectively mapped on $P_1, \ldots, P_{2n+3}$
- for all $k \in \{1, \ldots, 2n\}$, one computation of size $(2 - 1/N)k$ mapped on processor P_{k+3}
- for all $k \in \{1, \ldots, n+1\}$, one communication of size 1 between P_2 and P_{2k+1}
- for all $k \in \{1, \ldots, n\}$, one communication of size 1 between P_1 and P_{2k+2}

We can check that we always have $4N \geqslant (2 - 1/N)k \geqslant 1$ is $1 \leqslant k \leqslant 2n$. Because of this, computations of size $4N$ will always be put at the beginning of the schedule by the algorithm. Then, we have computations of size $(2-1/N)k$ and then the communications of size 1.

For example, the schedule of $H_{2*3}(N)$ looks like Figure 30.

For both $H_{2n}(N)$ et $H_{2n+1}(N)$, $0 \leqslant n \leqslant N$, points (1), (2), (3) and (6) are obvious.

Point (4) is very easy: The first communication of size $4N$ on P_1 is bigger than the gaps of P_1 (of size $(3 - 1/N)$). Because of this, the period of the schedule is bigger than K_{2n} (resp. $K_{2n+1}(N)$). Moreover, there is no task in the schedule after $t = K_{2n}(N)$ (resp. $t = K_{2n+1}(N)$), so K_{2n} (resp. $K_{2n+1}(N)$) is a period of the schedule. This leads to say that K_{2n} (resp. $K_{2n+1}(N)$) is the minimal period of the schedule of $H_{2n}(N)$ (resp. $H_{2n+1}(N)$).

For point (5) in $H_{2n}(N)$ (resp. $H_{2n+1}(N)$), the idea is to construct a schedule with

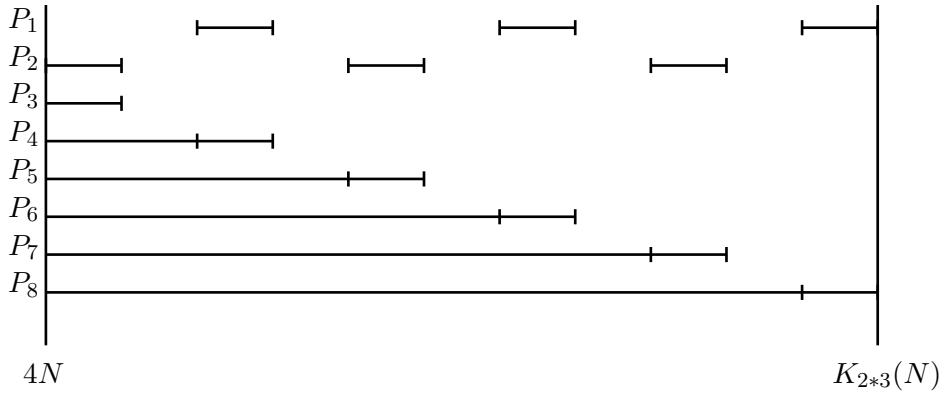


Figure 30: Representation of the schedule of $H_{2*3}(N)$. Computations of size $4N$ at the beginning of the schedule are not represented.

- all the computations of size $4N$ in the interval of time $[0, 4N]$
- all the communications of size 1 in the interval of time $[4N, 4N+n]$ (resp. $[4N, 4N+n+1]$)
- any computation of size $(2-1/N)k$ in the interval of time $[4N+n, 4N+n+(2-1/N)k]$ (resp. $[4N+n, 4N+n+1+(2-1/N)k]$)

For example, for $H_{2*3}(N)$, this second schedule looks like Figure 31.

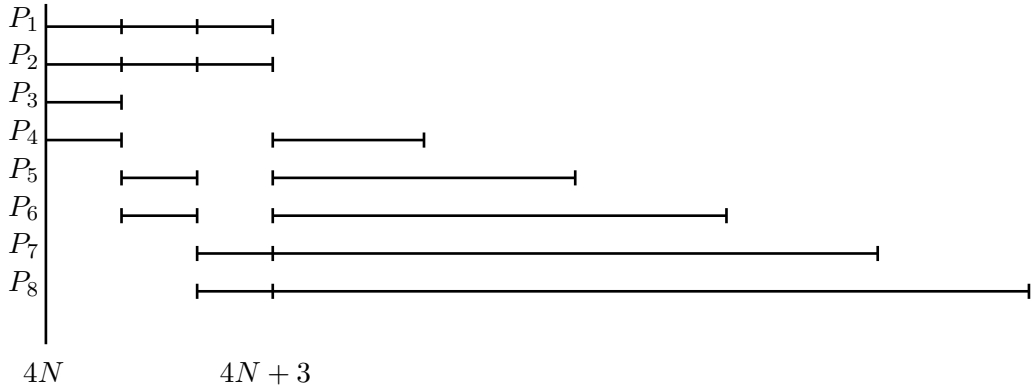


Figure 31: Representation of the second schedule of $H_{2*3}(N)$. Computations of size $4N$ at the beginning of the schedule are not represented.

This schedule has a period smaller than $4N+n+(2-1/N)(2n-1)$ (resp. $4N+n+1+(2-1/N)(2n)$). In both cases, this schedule has a period smaller than $8N+n$, which prove that point (5) is verified.

Induction

Let n be in $\mathbb{N}$, $n \geqslant N$. We suppose that we have $\{H_k(N)\}_{0 \leqslant k \leqslant 2n+1}$ (this is true if $n = N$, we saw it in the previous part). We wanted to show that we have $\{H_k(N)\}_{0 \leqslant k \leqslant 2(n+1)+1}$. We will first prove that $\{H_k(N)\}_{0 \leqslant k \leqslant 2n+1}$ implies $H_{2n+2}(N)$, and then that $\{H_k(N)\}_{0 \leqslant k \leqslant 2n+2}$ implies $H_{2n+3}(N)$.

We have $\{H_k(N)\}_{0 \leqslant k \leqslant 2n+1}$, in particular, $H_{2(n-N)+2}(N)$ and $H_{2n+1}(N)$ are true. The platform using for $H_{2n+1}(N)$ consists of $\alpha_{2n+1}(N)$ processors $\{P_1, \ldots, P_{\alpha_{2n+1}}(N)\}$, and the platform for $H_{2(n-N)+2}(N)$ consists of processors $\{P'_1, \ldots, P'_{\alpha_{2(n-N)+2}}(N)\}$. We call $G(N)$ the graph obtained when concatenating graphs $G_{2(n-N)+2}(N)$ and $G_{2n+1}(N)$. The schedule obtained by the last algorithm looks like Figure 32.

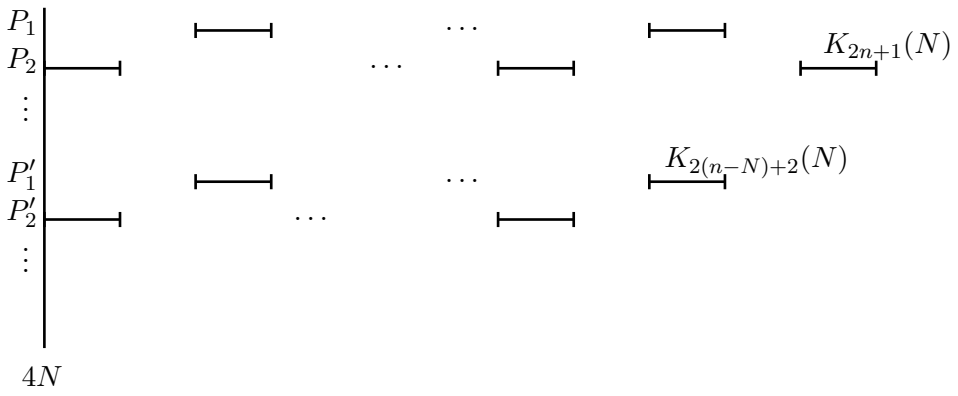


Figure 32: Schedule computed by the Longest First algorithm for graph $G(N)$, concatenation of $G_{2(n-N)+2}(N)$ and $G_{2n+1}(N)$.

We add to $G(N)$ $(4N-3)$ communications of size 1 between P'_1 and P'_2, Common gaps between P'_1 and P'_2 are of size $1 - 1/N$, so the algorithm put them at the end of the schedule. This is represented in Figure 33.

We also add to $G(N)$ one communication of size 1 between P_1 and P'_2. We call $G'(N)$ this graph. The corresponding schedule is represented in Figure 34.

We now claim that $H_{2n+2}(N)$ is true, with $G_{2n+2}(N) = G'(N)$. Points (1), (2), (3) and (6) are obvious by construction. The minimal period of the new schedule is $K_{2n+2}(N)$ because the last task on P_1 ends at $t = K_{2n+2}(N)$, all tasks ends before $K_{2n+2}(N)$ by construction. Because of this, the minimal period of the schedule is smaller than $K_{2n+2}(N)$. Moreover, there is no gap of size bigger than 3 in P_1, and there is a computation of size $4N > 3$ on P_1, so the minimal period is also bigger than $K_{2n+2}(N)$, which prove point (4).

We know that there exists a schedule for the graph $G_{2(n-N)+2}(N)$ such that all tasks on P'_1 and P'_2 are set in the interval of time $[0, 4N + n - N + 1]$, and such that the period is smaller than $8N + n - N + 1$. We also know that there exists a schedule for the graph $G_{2n+1}(N)$ such that all tasks on P_1 are set in the interval of time $[0, 4N + n]$ and all tasks on P_2 are set in the interval of time $[0, 4N + n + 1]$, and such that the period of this schedule is smaller than $8N + n$.

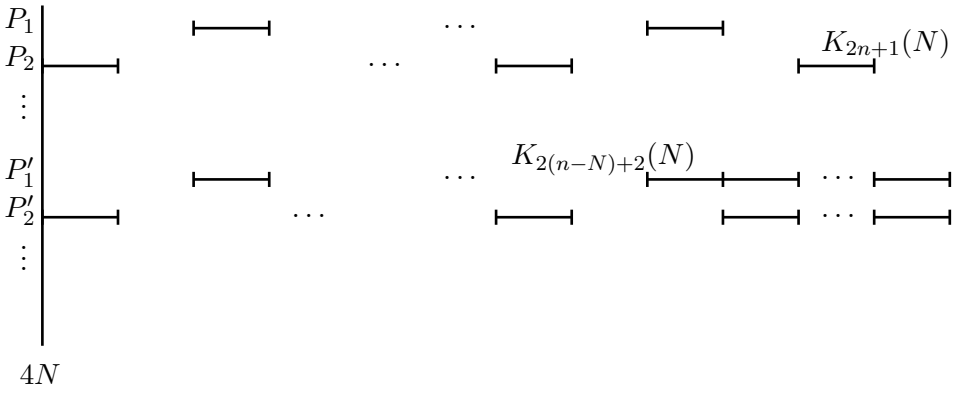


Figure 33: Schedule computed by the Longest First algorithm for graph $G(N)$, concatenation of $G_{2(n-N)+2}(N)$ and $G_{2n+1}(N)$, and for $4N-3$ communications of size 1 between P'_1 and P'_2.

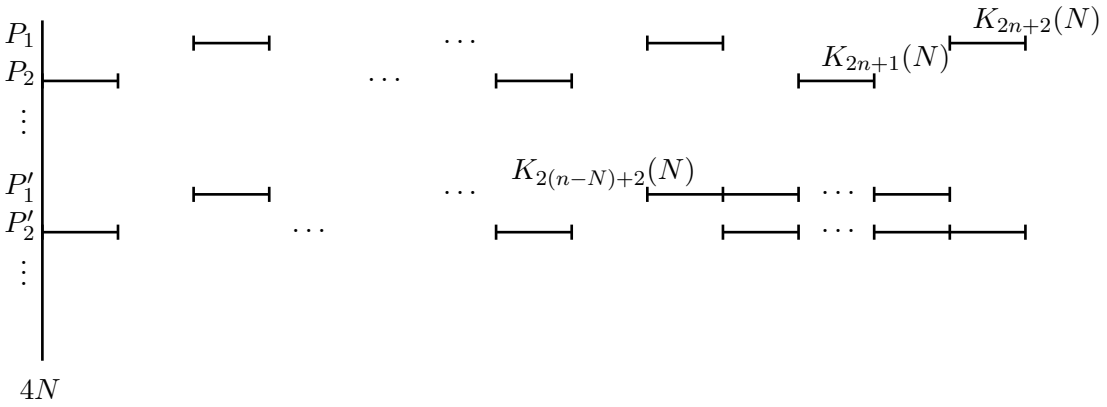


Figure 34: Schedule computed by the Longest First algorithm for graph $G'(N)$.

This means there exists a schedule for $G(N)$ which looks like Figure 35.

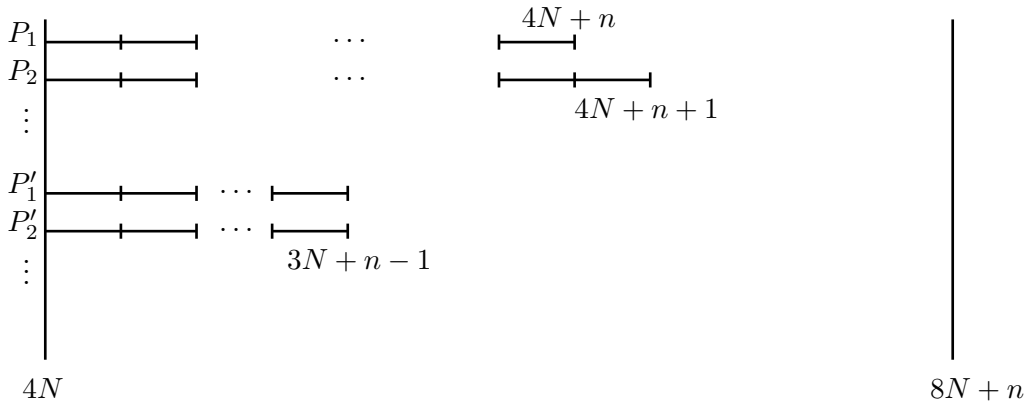


Figure 35: A schedule of small period for the graph $G(N)$.

We can add the communication between P_1 and P'_2 at $t = 4N + n$ (this is necessary to respect (5) and communications between P'_1 and P'_2 between $t = 4N+n+1$ and $t = 8N+n-2$. The corresponding schedule is represented in Figure 36. This proves that we respect point (5),

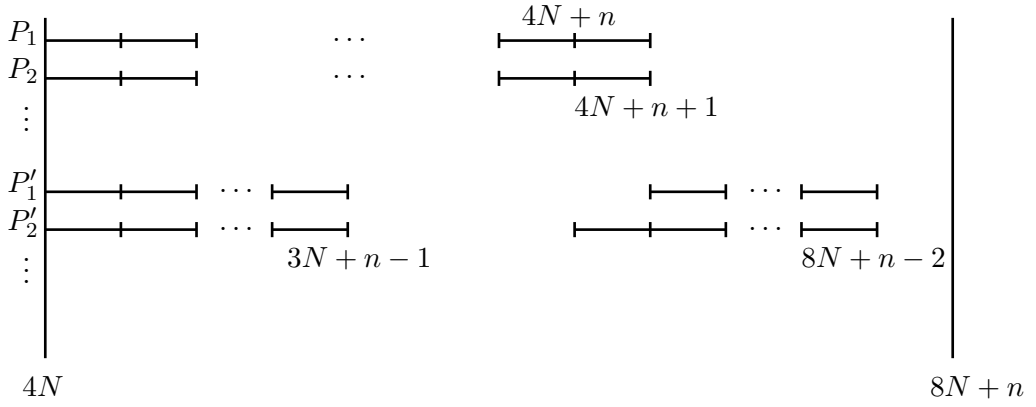


Figure 36: A schedule of small period for the graph $G'(N)$.

and ends the first par of the proof: $\{H_k(N)\}_{0 \leqslant k \leqslant 2n+1}$ implies $H_{2n+2}(N)$. The construction which proves that $\{H_k(N)\}_{0 \leqslant k \leqslant 2n+2}$ implies $H_{2n+3}(N)$ is similar. Because of this, we do not report it here. This ends the induction.

The approximation problem

Let us come back to the initial problem. We previously shown that, for any $(N, n) \in \mathbb{N}^2$, with $N \geqslant 2$, ones can construct a linear graph such that the schedule computed by the last algorithm has a minimal period $K_{2n}(N)$ with

$$K_{2n}(N) = 4N + 4n - 1 - \frac{2n-1}{N}$$

and such that the optimal period $K_{\min}(n, N)$ follows

$$K_{\min}(n, N) \leqslant 8N + n$$

We have

$$\lim_{n\to\infty} \frac{K_{2n}(N)}{8N+n} = 4 - 2/N$$

which means that for any $\varepsilon > 0$, there exists $n_1(N, \varepsilon)$ such that

$$\frac{K_{2n_1}(N)}{8N+n_1} \geqslant 4 - 2/N - \varepsilon$$

We can choose $\varepsilon = 1/N$. There exists $n_1(N, 1/N)$ such that

$$\frac{K_{2n_1}(N)}{8N+n_1} \geqslant 4 - 3/N$$

i.e.

$$K_{2n_1}(N) \geqslant (4 - 3/N)(8N + n1) \geqslant (4 - 3/N)K_{\min}(n1, N)$$

At the beginning of this proof, we let $\beta < 4$. There exists N_β such that $4 - 3/N_\beta > \beta$, and there exists a linear graph and a mapping such that the Longest First algorithm constructs a schedule of period $K_{2n_1}(N_\beta)$, with $K_{2n_1}(N_\beta) > \beta(8N_\beta + n_1) \geqslant \beta K_{\min}$, which ends the proof. □

4.3 Bi-criteria

Proposition 4.8. *Given a linear workflow and a mapping, the problem of computing the operation list that leads to a given period K and a given latency L is NP-hard in the one-port model without overlap.*

Proof. By Theorem 4.2, we showed that the problem of computing the operation list that leads to a given period K is NP-hard. Therefore, the bi-criteria problem also is NP-hard. □

5 Finding the optimal schedule for a given mapping in the multi-port model

5.1 Model

Let us recall that in the multi-port model, we assume that communications along different links can take place simultaneously. Because we deal with processor network cards, we bound the total communication capacity of each network card: we denote by B_u^i (resp. B_u^o) the capacity of the input (resp. output) network card of processor P_u. P_u cannot receive more than $1/B_u^i$ data items per time-unit, and it cannot send more than $1/B_u^o$ data items per time-unit.

We also consider that computations and communications can be done in parallel (overlap model), the input for data set $i + 1$ can be received while computing for data set i and sending result for data set $i - 1$.

5.2 Latency

Computing the schedule of optimal latency in the multi-port model with overlap is done exactly as in the one-port model without overlap. The equivalent of Theorem 4.1 is:

Theorem 5.1. *Given a linear workflow and a mapping, the problem of computing the schedule that leads to the optimal latency has polynomial complexity in the multi-port model with overlap.*

Proof. Because we just deal with latency, we can suppose that the period is long enough to separate the computation of different data sets in the linear workflow. This way, the optimal order is obvious: we do all the computations and all the communications as soon as possible. The corresponding latency is the sum of all computation and communication times. □

5.3 Period

Complexities of period minimization differ if we use the one-port model without overlap or the multi-port model with overlap. Theorem 4.2 states that finding a schedule of optimal period is NP-hard in the one-port model without overlap. On the contrary, we now show that this is easy in the multi-port model with overlap.

Theorem 5.2. *Given a linear workflow and a mapping, the problem of computing the schedule that leads to the optimal period $K_{\min}$ has polynomial complexity with the multi-port model with overlap. Moreover, $K_{\min} = CT$, where CT is the cycle-time.*

Proof. We basically process in two stages: firstly, we recall that the cycle-time CT is a lower bound of the optimal period K_{min}. Then, we compute in polynomial time a schedule who respects all the constraints and that has a period $K = CT$.

For any processor P_k, the minimum amount of time-units needed for the computations of one data-set is given by $\frac{1}{s_k}\sum_{a(i)=k} w_i$. This bound is reached when the processor P_k computes everything at maximal speed and without pause. Because of this, for any schedule, the period K follows

$$K \geqslant \max_{k\in\{1,\ldots,p\}} \left\{ \frac{1}{s_k} \sum_{a(i)=k} w_i \right\} \tag{21}$$

Similarly, we can bound the period by the minimum amount of time-units needed for the communications between two processors p_k and $p_{k'\neq k}$, for the input communication of every network card, and for the output communication of every network card. Combining all these bounds leads to $K \geqslant Q$, with :

$$Q = \max \begin{cases} \max\limits_{k\in\{1,\ldots,p\}} \left\{ \dfrac{1}{s_k} \displaystyle\sum_{a(i)=k} w_i \right\} \\ \max\limits_{k,k'\in\{1,\ldots,p\}\cup\{in,out\},k\neq k'} \left\{ \dfrac{1}{b_{k,k'}} \displaystyle\sum_{a(i)=k,a(i+1)=k'} \delta_i \right\} \\ \max\limits_{k\in\{1,\ldots,p\}\cup\{in\}} \left\{ \dfrac{1}{B_k^o} \displaystyle\sum_{a(i)=k} \delta_i \right\} \\ \max\limits_{k\in\{1,\ldots,p\}\cup\{out\}} \left\{ \dfrac{1}{B_k^i} \displaystyle\sum_{a(i)=k} \delta_{i-1} \right\} \end{cases} \tag{22}$$

and ones can check that $Q = CT$, by definition of the cycle-time CT_u of a processor P_u (see equation 3) and by definition of the cycle-time CT (see equation 4).

We now give a schedule who leads to a period $K = CT$ and which is computable in polynomial time. We assume that data-sets come at $t_0 + k.CT$, with $k \in \mathbb{Z}$. For each data-set, any processor P_u has three kind of tasks to do : some computations for stages $\{S_i\}_{a(i)=u}$, some input communications between stages $\{S_{i-1}, S_i\}_{a(i)=u}$ and some output communications between stages $\{S_i, S_{i+1}\}_{a(i)=u}$. We describe how theses tasks are computed in parallel in the interval of time $t_0, t_0 + CT$ for some data-sets. By periodicity, this describes all the schedule.

Input communications are all done in parallel. They begin at t_0, end at $t_0 + CT$, and any communication between stages S_{i-1} and S_i uses of bandwidth of δ_{i-1}/CT. Similarly, output communications are done in parallel, and any communication between S_i and S_{i+1} uses a bandwidth of δ_i/CT. All communications between processors are perfectly synchronous, and if a processor P_{u_1} sends a data of size δ_i using a bandwidth of δ_i/CT to a processor P_{u_2}, we verify that processor P_{u_2} receives a data of size δ_i from P_{u_1} using a bandwidth of δ_i/CT. Moreover, the bandwidth used between processors P_{u_1} and P_{u_2} is $\sum_{a(i)=u_1, a(i+1)=u_2} \delta_i/CT$. By definition of CT, we have

$$\sum_{a(i)=u_1, a(i+1)=u_2} \delta_i/CT \leqslant \left(\sum_{a(i)=u_1, a(i+1)=u_2} \delta_i \right) \times \frac{b_{u_1,u_2}}{\sum_{a(i)=u_1, a(i+1)=u_2} \delta_i} = b_{u_1,u_2}$$

and the bandwidth b_{u_1,u_2} between processors P_{u_1} and P_{u_2} is respected. Similarly, we check that bandwidths of network cards B_u^i and B_u^o are respected.

The schedule for computations of stages $\{S_i\}_{a(i)=u}$ is simpler: P_u executes the computations $\{w_i\}_{a(i)=u}$ one per one, from time t_0 to t_1. Processor P_u has a speed s_u, so

$$t_1 = t_0 + \sum_{a(i)=u} w_i/s_u \leqslant t_0 + CT$$

by definition of CT. This shows that P_u can make the computations $\{w_i\}_{a(i)=u}$ between t_0 and $t_0 + CT$, and concludes the proof.

□

5.4 Bi-criteria

In the multi-port model with overlap, complexities of period minimization and latency minimization are both polynomial. Because of this, we now address the problem of computing a schedule respecting both a period and a latency. We first deal with period minimization at minimal latency.

5.4.1 Minimizing period at minimal latency is polynomial

In this part, we prove that minimizing the period at minimal latency in the multi-port model is polynomial. Obviously, this does not prove that the bi-criteria problem (which consists in finding a schedule that respects a given period and a given latency) is polynomial. We recall that finding the minimal latency is polynomial (see Theorem 5.1).

Theorem 5.3. *Given a linear workflow and a mapping, the problem of computing the schedule that minimize the period at minimal latency $L_{\min}$ has polynomial complexity in the multi-port model.*

Proof. We recall that computing the minimal latency $L_{\min}$ in the multi-port model has a polynomial complexity. Moreover, finding the unique first data set schedule which corresponds to that latency has also a polynomial complexity.

Lemma 5.4. *In the multi-port model and at minimal latency $L_{\min}$, the minimal period $K_{\min|\ L_{\min}}$ follows*

$$\frac{L_{\min}}{p^2} \leqslant K_{\min|\ L_{\min}} \leqslant L_{\min}$$

Proof. First of all, let us recall that the minimal latency follows

$$L_{\min} = A + B$$

with

$$A = \sum_{u=1}^{p} \sum_{a(i)=u} \frac{w_i}{s_u}$$

$$B = \sum_{u=1}^{p} \sum_{v=1, v\neq u}^{p} \sum_{a(i)=u, a(i+1)=v} \frac{\delta_i}{\min(b_{u,v}, B_v^i, B_u^o)}$$

which traduces that $L_{\min}$ is the sum of all the computations (A) and all the communications (B) at maximum speed.

Let us suppose that $A \geqslant \frac{L_{min}}{p}$. For any processor P_u, the minimal period $K_{\min|L_{\min}}$ is bigger than the computation time of processor P_u:

$$K_{\min|L_{\min}} \geqslant \sum_{a(k)=u} \frac{w_k}{s_u}$$

and this leads to

$$pK_{\min|L_{\min}} \geqslant \sum_{u=1}^{p} \sum_{a(i)=u} \frac{w_i}{s_u} = A$$

and by hypothesis we have $A \geqslant \frac{L_{min}}{p}$. Combining these equation follows

$$K_{\min|L_{\min}} \geqslant \frac{L_{min}}{p^2}$$

Let us now suppose that $A \leqslant \frac{L_{min}}{p}$, which means $B \geqslant L_{min}(1 - \frac{1}{p})$. For any couple of different processors (P_u, P_v), the minimal period is bigger that the communications time from processor P_u to processor P_v:

$$K_{\min|L_{\min}} \geqslant \sum_{a(i)=u, a(i+1)=v} \frac{\delta_i}{\min(b_{u,v}, B_v^i, B_u^o)}$$

This follows

$$p(p-1)K_{\min|L_{\min}} \geqslant B \geqslant L_{min}(1 - \frac{1}{p})$$

which can be rewritten

$$K_{\min|\ L_{\min}} \geqslant \frac{L_{min}}{p^2}$$

Moreover, this is obvious that $K_{\min|L_{\min}} \leqslant L_{\min}$, which ends lemma's proof.

□

Schedule representation

A schedule is entirely represented by the computation of one data set and by its period. We are working at minimal latency. Because of this, the (unique) schedule of the first data set is computable in polynomial time. We suppose that this schedule is represented as follows: for each processor $P_u, 1 \leqslant u \leqslant p$, we have five lists c (computations), b_{in} (input links bandwidth), b_{out} (output links bandwidth), B_{in} (input card bandwidth) and B_{out} (output card bandwidth) of elements (t_1, t_2, x). An element (t_1, t_2, x) in c (resp. b_{in}, b_{out}, B_{in}, B_{out}) represent the fact that the processor P_u is computing (resp. communicating), between t_1 and t_2 and use x per cent of its computing capacity (resp. input bandwith, output bandwidth, input card bandwidth, output card bandwidth). Note that in the list c, for any task (t_1, t_2, x) we have $x = 100\%$, because processors are always computing at full speed.

When we construct these $5p$ lists, we add, for each computation w_i, one element in c, and for each communication δ_i, four elements in b_{in}, b_{out}, B_{in}, B_{out} respectively. We can claim that the number of elements in a list is smaller than $n+1$, when n is the size of the linear graph.

Period of a list

We say that a list $l = (t_{1,i}, t_{2,i}, x_i)_i$ is K periodic if for any time $t \in [0, L_{\min}]$ we have

$$\sum_{\alpha \in \mathbb{Z}, \alpha K + t_{1,i} < t < \alpha K + t_{2,i}} x_i \leqslant 100$$

which means that the union of the lists $l = (t_{1,i} + \alpha K, t_{2,i} + \alpha K, x_i)_i$ for $\alpha \in \mathbb{Z}$ verifies that for any time t the sum of percentages of utilization x_i don't exceed 100%.

Period of a schedule

A schedule is of period K if, when we started to compute data sets each K time units, for any time t, all bandwidths are respected and if there is 0 or 1 computation on processor P_i. This is equivalent to say that the $5p$ lists are K periodic.

We now proceed as follows: we first show that for any list l of the $5p$ lists we can compute in polynomial time the domain of possible values for the period K in $[L_{\min}/p^2, L_{\min}]$. Then, we explain how to compute the intersection of these $5p$ domains, which is the domain of possible periods for the schedule included in $[L_{\min}/p^2, L_{\min}]$ at minimal latency. This domain has a minimum, which can be computed in polynomial time, and which is the period $K_{\min | \ L_{\min}}$.

Domain of periods $K \in [L_{\min}/p^2, L_{\min}]$ of a list l

Let $l = (t_{1,i}, t_{2,i}, x_i)_i$ be one of the $5p$ lists. l contains in most n elements. Our goal is to compute the domain I such that

$$K \in I \Longleftrightarrow K \text{ is in } [L_{\min}/p^2, L_{\min}] \text{ and } l \text{ is } K \text{ periodic}$$

A value K of $[L_{\min}/p^2, L_{\min}]$ is in I if for any time $t \in [0, L_{\min}]$ we have

$$\sum_{\alpha \in \mathbb{Z}, \alpha K + t_{1,i} < t < \alpha K + t_{2,i}} x_i \leqslant 100 \tag{23}$$

We know that $t_{1,i}$ and $t_{2,i}$ are in $[0, L_{\min}]$. Because of this, we can restrict the domain of α to

$$[-\frac{L_{\min}}{L_{\min}/p^2} - 1, \frac{L_{\min}}{L_{\min}/p^2}] \bigcap \mathbb{Z}$$

which can be rewritten

$$[-p^2 - 1, p^2] \bigcap \mathbb{Z} = I_\alpha$$

We can rewrite equation 23:

$$\forall t \in [0, L_{\min}], \quad \sum_{\alpha \in I_\alpha, \alpha K + t_{1,i} < t < \alpha K + t_{2,i}} x_i \leqslant 100 \tag{24}$$

When K is fixed, there is a finite number C of bounds $\alpha K + t_{1,i}$ and $\alpha K + t_{2,i}$. Let us suppose that these bounds are represented by a list $(x_j(K))_{1 \leqslant j \leqslant C}$ with $\forall\ j \in \{1, \ldots, C-1\},\ x_j(K) \leqslant x_{j+1}(K)$. Let ε be in $\mathbb{R}_+^*$. Let us suppose that we have

$$\begin{aligned} \forall\ 1 \leqslant j \leqslant C-1, &\quad x_j(K-\varepsilon) \leqslant x_{j+1}(K-\varepsilon) \\ \forall\ 1 \leqslant j \leqslant C, &\quad x_j(K-\varepsilon) \leqslant 0 \iff x_j(K) \leqslant 0 \\ \forall\ 1 \leqslant j \leqslant C, &\quad x_j(K-\varepsilon) \leqslant L_{\min} \iff x_j(K) \leqslant L_{\min} \end{aligned}$$

$\{x_j\}_j$ are affine functions of K, so we also have, for all $e \in [0, \varepsilon]$

$$\begin{aligned} \forall\ 1 \leqslant j \leqslant C-1, &\quad x_j(K-e) \leqslant x_{j+1}(K-e) \\ \forall\ 1 \leqslant j \leqslant C, &\quad x_j(K-e) \leqslant 0 \iff x_j(K) \leqslant 0 \\ \forall\ 1 \leqslant j \leqslant C, &\quad x_j(K-e) \leqslant L_{\min} \iff x_j(K) \leqslant L_{\min} \end{aligned}$$

This leads to say that, for all $e \in [0, \varepsilon]$,

$$\forall t \in [0, L_{\min}], \quad \sum_{\alpha \in I_\alpha, \alpha(K-e) + t_{1,i} < t < \alpha(K-e) + t_{2,i}} x_i \leqslant 100 \tag{25}$$

i.e. for all $e \in [0, \varepsilon]$, the list l is $(K-e)$-periodic. We can do the same remark if l is not K-periodic.

This remark permits to construct an algorithm to compute the domain I of periods of the list l in $[L_{\min}/p^2, L_{\min}]$. We can compute all the critical values of K such that $\alpha K + t_{1/2,i} = \alpha' K + t_{1/2,j}$, all the critical values such that $\alpha K + t_{1/2,i} = 0$ (we suppose that this function is not the null function), and all the critical values such that $\alpha K + t_{1/2,i} = L_{\min}$ (we suppose that this function is not the constant function equal to $L_{\min}$). There is a polynomial number D of critical values (the number is $O(p^4 n^2)$), and it takes a polynomial time in n and p to compute them and to put them in an ordered list $K_0 < K_1 < K_2 < \ldots < K_D$ (we don't put a same value twice is the list).

Then, for each couple (K_i, K_{i+1}), we can test if l is $\frac{K_i + K_{i+1}}{2}$ periodic. If yes, we add $[K_i, K_{i+1}]$ to I. Else, we know that there is no value K in $]K_i, K_{i+1}[$ such that the list l is K-periodic, so we add K_i if the list is K_i periodic, K_{i+1} is the list is K_{i+1} periodic, and nothing else. All of this permits to compute I in polynomial time and I is represented by a union of $O(p^4 n^2)$ intervals.

How to compute the final period $K_{\min\,|L_{\min}}$?

The minimal period of the schedule $K_{\min}$ is the minimal common period of the $5p$ lists c, b_{in}, b_{out}, B_{in} and B_{out}. Because of lemma 7, we know that

$$\frac{L_{\min}}{p^2} \leqslant K_{\min} \leqslant L_{\min}$$

so the minimal period of the schedule is also the minimal common period of the $5p$ lists intersected by $[L_{\min}/p^2, L_{\min}]$. We showed that these domains $(I_k)_{1 \leqslant k \leqslant 5p}$ are computable in polynomial time and are of size $O(p^4 n^2)$. We can compute the intersection of I_1 and I_2, then $I_1 \cup I_2$ and I_3, etc. This has a complexity $O(p^5 n^2)$. The minimal period $K_{\min}$ is the minimum number of the intersection $I_1 \cup \ldots \cup I_{5p}$ and is computable in polynomial time, since this domain is represented by a sum of $O(p^5 n^2)$ closed intervals. This ends the proof. □

5.4.2 Minimizing latency at minimal period is NP-hard

It the previous section, we explain why minimizing period at minimal latency has polynomial complexity in the multi-port model when the mapping is given. We now prove that the opposite problem is NP-hard.

Theorem 5.5. *Given a linear workflow and a mapping, the problem of computing a schedule that minimizes latency at minimal period $K_{\min}$ is NP-hard in the weak sense in the multi-port model with overlap. The corresponding latency is called $L_{min|K_{\min}}$.*

Proof. We consider the associate decision problem and show that it is NP-complete: given a linear graph, a mapping, and a bound L, does there exists a schedule such that the period is the minimal period $K_{\min}$[8] and the latency does not exceed L? This problem is obviously NP: given a linear graph, a mapping and a schedule, we can compute the minimal period $K_{\min}$ in polynomial time (see Theorem 5.2), and check that the period of the schedule is $K_{\min}$ and the latency does not exceed L. To establish the completeness, we use a reduction from 2-PARTITION [7] that is NP-complete in the weak sense. As in the proof of Theorem 4.2, we consider an instance $\mathcal{I}_1$ of this problem: given a list of positive integers $(a_i)_{1 \leqslant i \leqslant n}$ such that $\sum_{i=1}^{n} a_i = K$, does there exist $\gamma \in \{1, \ldots, n\}$ such that:

$$\sum_{i \in \gamma} a_i = \sum_{i \notin \gamma} a_i = K/2 \tag{26}$$

We associate to $\mathcal{I}_1$ an instance $\mathcal{I}_2$ with $2n+5$ stages and 3 processors $\{P_1, P_2, P_3\}$, given by the linear graph and the mapping represented on Figure 37. We assume that the platform is fully homogeneous and the common computation speed is $s = 1$. All bandwidths are equal to 1[9].

Finally, we let $L = (n+2)K$. The size of $\mathcal{I}_2$ is obviously linear in the size of $\mathcal{I}_1$. The minimal period of this linear graph and mapping is $K_{\min} = K$.

Suppose first that $\mathcal{I}_1$ has a solution, i.e. there exists $\gamma \in \{1, \ldots, n\}$ such that

$$\sum_{i \in \gamma} a_i = \sum_{i \notin \gamma} a_i = K/2$$

We construct a schedule of period K as represented on Figure 38. The computation of one data set is represented in bold. The key in this schedule is that the communication between stages S_{2n+2} and S_{2n+3}, the computation on S_{2n+3} and the communication between stages S_{2n+3} and

[8] In Theorem 5.2, we saw that finding the minimal period $K_{\min}$ can be done in polynomial time.

[9] In fact, it is not necessary to precise bandwidths because all communications are of size 0.

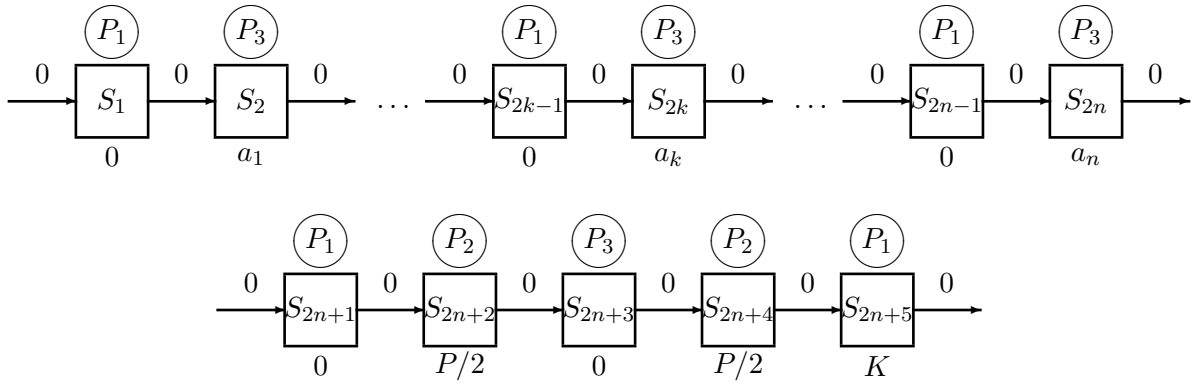


Figure 37: Representation of the instance $\mathcal{I}_2$ which consists of a linear graph with $2n+5$ stages mapped on 3 processors.

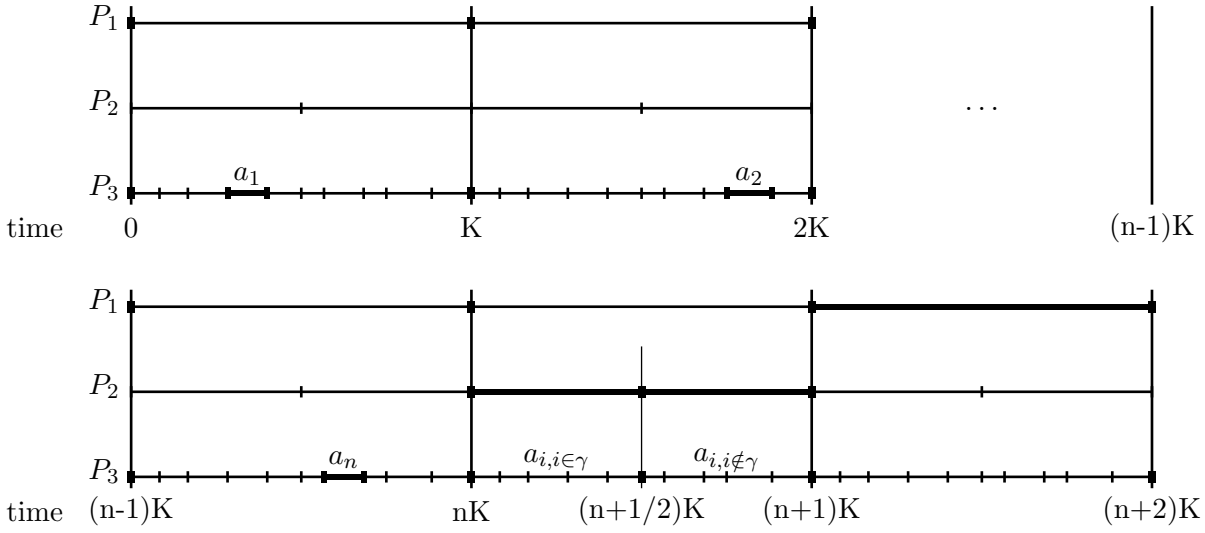


Figure 38: Representation of a schedule of period K, for the linear graph and the mapping given by Figure 37. The computation of one data set is represented in bold. The latency is $(n+2)K = L$.

S_{2n+4} occur at time $t = (n + 1/2)K$. This is possible because computations of $\{a_i\}_{i\in[\![1,n]\!]}$ are "2-PARTITIONED", which means there is no computation at time $t = (n + 1/2)K$, and the computation of size 0 of stage S_{2n+3} do not have to wait before being executed, and the latency $L = (n + 2)K$ is respected.

We now suppose that $\mathcal{I}_2$ has a solution. We basically want to prove that the computation of stage S_{2n+3} has to occur at time $t = (n + 1/2)K$ for respecting the latency L. This computation requires a 2-PARTITION of the computations of size a_i on P_3. Let us suppose that for one data set the computation of stage S_{2n+5} begin on P_1 at time $t = 0$. This computation is done each K time units, which means that P_1 is computing for stage S_{2n+5} all the time, and a computation of size 0 on P_1 can only occur at time $\alpha K, \alpha \in \mathbb{Z}$. There is a data set d such that the computation of stage S_1 of size 0 on P_1 is done at $t = 0$. For stage S_2 there is a computation of size $a_1 > 0$, and computation of stage S_3 can not occur before $t = K$. Similarly, there is for stage S_3 a computation of size $a_2 > 0$ on P_1 and computation of stage S_4 occur after $t = nK$. The sum of computations of stages S_{2n+1} to S_{2n+5} is $2K$ so the output computation occur after $t = (n+2)K$, which means that the latency is bigger than $(n+2)K = L$.

We assumed that the latency is smaller than L, so we know that the latency is exactly L and that computation of stages S_{2n+1} to S_{2n+5} for data set d occur in the interval of time $[nK, (n + 2)K]$, exactly as in Figure 38. This proves that computation of stage S_{2n+3} of size 0 occur on P_3 at time $t = (n + 1/2)K$, and computations for stages $S_{2k-1}, k \in [\![1, n]\!]$ occurring in the interval of time $[nK, (n + 1)K]$ occur in the interval of time $[nK, (n + 1/2)K]$ or in the interval of time $[(n + 1/2)K, (n + 1)K]$. There is no idle time on P_3, and this constructs a 2-PARTITION of integers $\{a_1, a_2, \ldots, a_n\}$, which ends the proof.

□

5.4.3 The bi-criteria problem

Theorem 5.6. *Given a linear workflow and a mapping, the problem of computing a schedule that respects a period K and a latency L is NP-hard in the weak sense, in the multi-port model with overlap.*

Proof. With Theorem 5.5, we now that the problem of computing a schedule that respects a period $K_{\min}$ and a latency L is NP-hard in the weak sense in the multi-port model with overlap. Therefore, the general case is also NP-hard in the weak sense. □

6 Conclusion

This work presents complexity results for finding optimal schedules for linear pipelined graphs, when the mapping is given, both for the one-port model without overlap and for the multiport model with overlap. We provided a formal definition of both models and various objectives: latency minimization, period minimization and the bi-criteria problem. Altogether, with three objectives and two models, we present six main complexity results. In both models, latency minimization is easy, whereas period minimization can be done in polynomial time in the multiport model with overlap and is NP-hard in the one-port model without overlap. Finally, for both models, the bi-criteria problem is NP-hard. We also provide an 4-approximation algorithm for period minimization in the one-port model, and we provide some relations between the cycle-time and the optimal period.

In the future, we plan to find approximations, or at least efficient heuristics, for the bi-criteria problem for both models. Moreover, theses results have to be extended to models that allow preemption. This would require to carefully assess the cost of interruptions. Some preliminary work has already be done in that sense, but more interesting problems (bi-criteria problems) are not yet solved.

References

[1] K. Agrawal, A. Benoit, and Y. Robert. Mapping linear workflows with computation/communication overlap. Research Report 2008-21, LIP, ENS Lyon, France, June 2008. Available at `graal.ens-lyon.fr/~abenoit/`.

[2] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Assessing the impact and limits of steady-state scheduling for mixed task and data parallelism on heterogenous platforms. Research Report RR-2004-20, LIP, ENS Lyon, France, Apr. 2004. Available at the url `http://graal.ens-lyon.fr/~yrobert`.

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

[4] P. B. Bhat, C. Raghavendra, and V. K. Prasanna. Efficient collective communication in distributed heterogeneous systems. In *Journal of Parallel and Distributed Computing*, pages 15–24. IEEE Computer Society Press, 1999.

[5] C. Consel, H. Hamdi, L. Réveillère, L. Singaravelu, H. Yu, and C. Pu. Spidle: a DSL approach to specifying streaming applications. In *Proc. 2nd Int. Conf. on Generative Programming and Component Engineering*, pages 1–17. Springer, 2003.

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

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

[8] J. Gummaraju, J. Coburn, Y. Turner, and M. Rosenblum. Streamware: programming general-purpose multicore processors using streams. In *Proc. 13th Int. Conf. on Architectural Support for Programming Languages and Operating Systems ASPLOS'2008*, pages 297–307. ACM Press, 2008.

[9] B. Hong and V. Prasanna. Bandwidth-aware resource allocation for heterogeneous computing systems to maximize throughput. In *Proceedings of the 32th International Conference on Parallel Processing (ICPP'2003)*. IEEE Computer Society Press, 2003.

[10] R. Newton, L. Girod, M. Craig, S. Madden, and G. Morrisett. Wavescript: A case-study in applying a distributed stream-processing language. Research Report MIT-CSAIL-TR-2008-005, MIT CSAIL, January 2008.

[11] R. Stephens. A survey of stream processing. *Acta Informatica*, 34(7):491–541, 1997.

[12] 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, PPoPP'95*, pages 134–143. ACM Press, 1995.

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

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

[15] W. Thies, M. Karczmarek, and S. Amarasinghe. Streamit: a language for streaming applications. In *Proceedings of 11th Int. Conf. on Compiler Construction*, LNCS 2304. Springer, 2002.

[16] N. Vydyanathan, U. Catalyurek, T. Kurc, P. Saddayappan, and J. Saltz. An approach for optimizing latency under throughput constraints for application workflows on clusters. Research Report OSU-CISRC-1/07-TR03, Ohio State University, Columbus, OH, Jan. 2007. Available at `ftp://ftp.cse.ohio-state.edu/pub/tech-report/2007/TR03.pdf`.

[17] N. Vydyanathan, U. Catalyurek, T. Kurc, P. Saddayappan, and J. Saltz. Optimizing latency and throughput of application workflows on clusters. Research Report OSU-CISRC-4/08-TR17, Ohio State University, Columbus, OH, Apr. 2008. Available at `ftp://ftp.cse.ohio-state.edu/pub/tech-report/2008/TR17.pdf`.