Scheduling Algorithms for Linear Workflow Optimization

K. Agrawal[1] A. Benoit[2] L. Magnan[2] Y. Robert[2]

[1] Washington University in St. Louis, USA [2] ENS Lyon et Université de Lyon, France
kunal@cse.wustl.edu,{Anne.Benoit|Loic.Magnan|Yves.Robert}@ens-lyon.fr

***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*, i.e., the order in which the various stages execute on their assigned processors. In this paper, we assume that the mapping is given and explore the latter problem of scheduling, particularly for *linear* workflows. Linear workflows are those in which dependencies between stages can be represented by a linear graph. The objective of the scheduling algorithm is either to minimize the period (the inverse of the throughput), or to minimize the latency (response time), or both. We consider two realistic execution models: the one-port model (all operations are serialized) and the multi-port model (bounded communication capacities and communication/computation overlap). In both models, finding a schedule to minimize the latency is easy. However, computing the schedule to minimize the period 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 the 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.**

Keywords: pipeline graphs, workflow, scheduling, mapping, period, throughput, latency, complexity results.

I. Introduction

Pipelined workflows are a popular paradigm for *streaming applications* like video and audio encoding and decoding, DSP applications, etc. [1]–[5]. Streaming applications are becoming increasingly prevalent, and many languages [6]–[9] 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 onto the various processors of the machine. After finding the mapping, we must compute the *schedule*, i.e., the order in which the various operations (computations and communications) execute on their assigned processors. Since data continually flows through workflow applications, the goal is often to find a mapping and a schedule that decreases the period and/or the latency [10]–[14]. The *period* of a schedule (which is the inverse of the throughput) is the time-interval between the input of two successive data sets, while the *latency*, or response time, is the time for each data set to complete its whole execution.

Workflows are usually represented by Directed Acyclic Graphs (DAGs for short). In the most general case where the DAG is arbitrary, most problems related to period and/or latency minimization are NP-hard. Vydynathan et al. consider bi-criteria minimization problems on homogeneous platforms. In [10], they show that finding the optimal mapping is NP-hard, and they give a mapping and scheduling heuristic. Their algorithm minimizes the latency while satisfying strict throughput requirements. In [11], they also describe a heuristic that optimizes the period 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 [2] and Beaumont et al. [15] 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 for a special class of DAGs, namely *linear graphs*. Subhlok and Vondran study the problem of mapping linear graphs on homogeneous platforms. In [12], they prove that minimizing the period has a time complexity $O(p^4n^2)$, when p is the number of processors and n the number of stages (nodes in the linear graph). In [13], they extend these results and show that finding the optimal latency while respecting a given period, also has a time complexity $O(p^4n^2)$. We point out that they assume that the number of processors p is larger than the number of tasks n, which is a strong assumption that we do not use in this paper. Without this assumption, minimizing the period is easily proved to be NP-hard, using a reduction from 2-PARTITION [16]. This work was extended to heterogeneous platforms in [14], [17].

In this paper, we make an even stronger assumption: while we still restrict to linear graphs, we assume that *the mapping is given*, and we explore the problem of *scheduling* the execution graph so as to optimize period and latency. It turns out that the difficulty of the latter scheduling problem has been underestimated so far in the literature. In fact, our paper [18] does perform a similar study, as it investigates the complexity of scheduling filtering applications once the mapping is given. However, the results of [18] are restricted to one-to-one mappings, and apply to general DAGs with filtering services. Here we investigate general mappings for linear workflow applications. To the best of our knowledge, this study is the first to assess the complexity of orchestrating computations and communications for linear workflows, once the mapping has been determined, although this is a very natural and important problem.

Indeed, one might ask the question, why do we care about solving the scheduling problem when the mapping problem (finding the mapping with minimum period or latency) is often already NP-complete. Even if the mapping problem is NP complete, the scheduling problem is an interesting problem in its own right. For example, for some problems, the mapping might be fixed due to other constraints like memory. As another example, in some problems, some stages may have *affinity* (i.e., they want to be on the same processor) or *anti-affinity* (i.e., they have to be mapped on different processors). For these problems the mapping may be decided due to these other constraints, but we still want to find the schedule that minimizes the period or latency or both.

The emphasis of the paper is on *realistic* communication models. The first model is the *one-port model* 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 for representing single-threaded systems. The second model is the *multi-port model*, which allows multiple incoming and outgoing communications to take place simultaneously as long as network card and bandwidth constraints are fulfilled. In this model, which corresponds to multi-threaded systems, we further assume that communications and computations can be overlapped.

The major contributions of the paper are the following:

- finding the optimal latency $L_{\min}$ is easy in both the one-port and multi-port models;
- finding the optimal period $K_{\min}$ is NP-hard in the one-port model, but has polynomial complexity in the multi-port model;
- enforcing a given period K and a given latency L is NP-hard in the multi-port model; this result is surprising because both period and latency minimization are polynomial for this model.

The rest of the paper is organized as follows. We first give some details about the framework (Section II). Section III is devoted to the one-port model, and shows that finding a schedule with optimal latency has polynomial complexity, while finding a schedule with optimal period is NP-hard. The section also presents a $4-$approximation algorithm for minimizing the period. Section IV is devoted to the multi-port model, and shows that finding a schedule with optimal latency and finding a schedule with optimal period can both be done in polynomial time. However, finding a schedule which respects both a given period and a given latency is NP-hard. Finally we offer some conclusions and directions of prospective future work in Section V.

II. Framework

This section is devoted to a precise statement of the different models and optimization problems.

A. Linear workflows

We restrict to linear workflows, whose dependence graphs are linear chains (see Figure 1). The 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 S_{k-1}, performs w_k computations and sends a data item of size δ_k to the next stage S_{k-1}. 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 results for data set $i-1$. Otherwise, these operations have to be done serially. We deal with both models, with and without communication/computation overlap (see below).

B. 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. Two special processors P_{in} and P_{out} are devoted to input/output data. There is a bidirectional link $link_{u,v} : P_u \leftrightarrow P_v$ between each processor pair (P_u, P_v) (where $u \neq v$), 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 time-units to execute X floating point operations on P_u. We account for the bounded capacity of processor network cards as follows: B^i_u (resp. B^o_u) represents the capacity of the input (resp. output) network card of processor P_u. P_u cannot receive more than $1/B^i_u$ data items per time-unit, and cannot send out more than $1/B^o_u$ data items per time-unit.

A platform is *fully homogeneous* if all processors are identical (they compute at the same speed s), and if all communication devices are identical: the bandwidth is the same (say b) between any processor pair, and all network cards have the same capacities (say B^i and B^o). The platform is *communication homogeneous* if communication devices are identical but processor speeds may differ. Finally, the platform is *fully heterogeneous* in the most general case. These platform categories are widely encountered. 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.

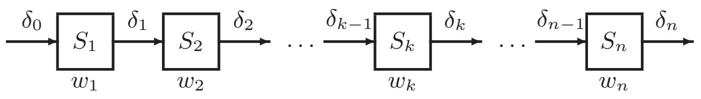

Figure 1: Representation of a linear workflow.

C. Communication model

We now provide a precise description of the communication model, since our results highly depend on this model. The standard model for DAG scheduling heuristics [19]–[21] does a poor job to model physical limits of interconnection networks. This 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 [22], where a processor can be involved in a single communication, either a send or a receive. However, independent communications between distinct processor pairs can take place simultaneously. For this model, we also consider that there is no overlap, i.e., a processor cannot simultaneously compute and communicate. This model does a good job for representing single-threaded systems.

Another realistic model is the *multi-port* model [23]. This model allows multiple simultaneous incoming (and outgoing) communications, but the total incoming (resp. outgoing) communication volume is bounded by the capacity of the network card. For this model, we assume that a processor can overlap independent computations and communications. This model is representative of current multi-threaded systems. In the following, we require that communications use a constant bandwidth throughout their execution[1]. This means that when a communication of size δ_i begins from processor P_u to processor P_v, it uses a constant bandwidth $b_i \leqslant \min\left(b_{u,v}, B_u^o, B_v^i\right)$ during δ_i/b_i time-steps.

D. Mapping

To execute the workflow, application stages S_k are assigned to platform processors P_u. This assignment is a *mapping*, and it is defined by an allocation function $a : [\![1,n]\!] \longrightarrow [\![1,p]\!]$. Here $a(k) = u$ means that the computation of stage S_k is executed by processor $P_u = P_{a(k)}$. We extend the domain of definition of a to $\{0,\dots,n+1\}$ by $a(0) = \mathit{in}$ and $a(n+1) = \mathit{out}$. This means $P_{a(0)} = P_{\mathit{in}}$ and $P_{a(n+1)} = P_{\mathit{out}}$, with P_{in} the input processor, and P_{out} the output processor.

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 2, where S_1 and S_3 are mapped on P_2, S_2 on P_3, and S_4 on P_1.

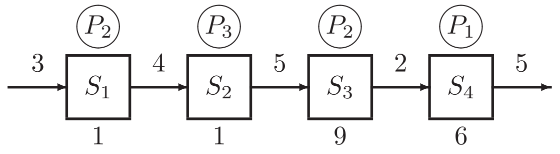

Figure 2: Representation of a linear graph and a mapping.

Given a linear graph G and a mapping a, we define the *cycle-time* $CT_u(a,G)$ of a processor P_u as the time it needs to process all the operations it has been assigned. The formal definition differs for the one-port and multi-port models. 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:

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

In the multi-port model, communications and computations can take place in parallel, but we have to account for all physical resource limitations:

$$\begin{aligned} CT_u(a,G) \quad &= \max \Bigg(\frac{1}{s_u} \sum_{a(i)=u} w_i, \\ & \max_{k\in\{1,\dots,p\}\cup\{\mathit{in},\mathit{out}\},k\neq u} \left\{ \frac{1}{b_{k,u}} \sum_{a(i)=k,a(i+1)=u} \delta_i \right\}, \\ & \frac{1}{B_u^i} \sum_{a(i-1)\neq u,\ a(i)=u} \delta_{i-1}, \\ & \max_{k\in\{1,\dots,p\}\cup\{\mathit{in},\mathit{out}\},k\neq u} \left\{ \frac{1}{b_{u,k}} \sum_{a(i)=u,a(i+1)=k} \delta_i \right\}, \\ & \frac{1}{B_u^o} \sum_{a(i)=u,\ a(i+1)\neq u} \delta_i . \end{aligned} \tag{2}$$

Finally, the cycle-time $CT(a,G)$ of the mapping is defined as:

$$CT(a,G) = \max\left\{CT_1(a,G),\dots,CT_p(a,G)\right\}. \tag{3}$$

It is important to point out that the cycle-time of each processor does not provide enough information to execute the workflow: even if we know the total amount of communications for each processor, we need to specify in which order these operations are executed.

E. Scheduling

For a given mapping, there exist different ways to execute the application on the platform. For example, suppose that the linear graph consists of two stages with $w_1 = 2, w_2 = 3, \delta_0 = \delta_2 = 0$ and $\delta_1 = 1$. Stage S_1 (resp. S_2) is mapped onto P_1 (resp. P_2), both processors are of speed $s_1 = s_2 = 1$ and communications also are homogeneous: $b = B^i = B^o = 1$.

Two different schedules of period $K = 4$ are represented on Figure 3, using the one-port model. For each schedule, the computation 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. Note 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 other words, the latency is equal to 7 for the first schedule and to 6 for the second one.

[1]Having a variable bandwidth might be a reasonable assumption when preemption is allowed, which is not the case here.

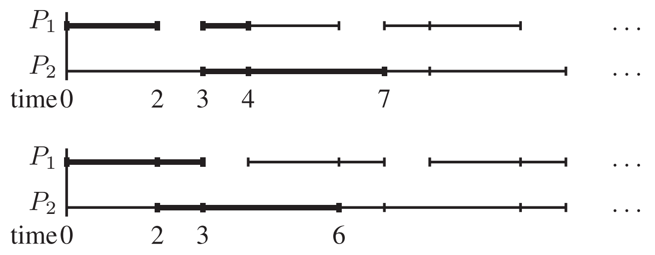

Figure 3: Two different schedules of period $K = 4$ for the same mapping. The computation of one data set is represented in bold.

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 fraction of the bandwidth: the non-used bandwidth cannot be useful for any other communication. Therefore, we assume that, in the one-port model, every communication occurs using the maximal bandwidth which fulfills link bandwidth and network card limitations. Formally, a communication from P_u to P_v uses a bandwidth b such that $b = \min\left\{b_{u,v}, B^i_v, B^o_u\right\}$. Similarly, we assume that a computation on P_u is done at maximal speed $s = s_u$.

In the multi-port model, different communications may occur at the same time, but (as noted before) we require that they use a constant bandwidth throughout execution. We also assume that two computations cannot occur simultaneously on a given processor: hence each computation on processor P_u always executes at full speed s_u.

In both models, a schedule is formally described by an operation list that describes the time-steps at which every computation and every communication takes place. We look for *cyclic* schedule that repeats every K time-steps, where K is the *period*: a new data set enters the platform every K time-steps, and follows exactly the same sequence of operations as the previous data set, but K steps later. This allows us to specify the operation list concisely. Indeed, a schedule is fully determined by the time at which each operation (computation or communication) begins and ends for the first data set, and by the value of the period K. Therefore, the description of the operation list is polynomial (actually, quadratic) in the number n of workflow nodes. In addition, we can check in polynomial time that the schedule is valid, i.e., that there is no resource conflict and that no processing speed, link bandwidth or network card capacity is exceeded.

It is easy to see that, for both models, the period K of any schedule has to be larger than or equal to the cycle-time $CT(a, G)$. 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 $CT(a, G)$. We point out that in some situations the optimal period $K_{\min}$ is strictly larger than the cycle-time $CT(a, G)$ (see Theorem 2 in Section III-B).

As for the *latency*, it is defined as the time needed to execute a single data set entirely. Formally, assume that the first operation for the first data set begins at time 0. Then the latency L is equal to the time at which the last operation for this data set terminates.

F. Objective function

We consider three important objective functions:

- (i) minimizing the period K, i.e., find $K_{\min}$;
- (ii) minimizing the latency L, i.e., find $L_{\min}$;
- (iii) finding a schedule which enforces a given period K and a given latency L (bi-criteria problem).

Of course, period and latency are antagonistic objectives, hence the added difficulty of the bi-criteria problem where good trade-offs are searched for.

III. One-port model

In this section, we explore the problem of period and/or latency minimization in the one-port model.

A. Latency

Theorem 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.*

Proof: To minimize the latency, we can consider schedules with periods long enough to serialize the processing of each data set and minimize the execution time, or makespan, when processing a unique data set. In other words, we delay the processing of the next data set until the current one is completely executed, this suppresses all resource conflicts (with such a strategy, the period is equal to the latency). The optimal ordering of the operations is then 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. ■

B. Period

In this section, we first prove that the period minimization problem is NP-hard. Then we discuss several heuristics and we provide a $4-$approximation algorithm to solve the problem.

1) Complexity:

Theorem 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.*

Proof: We consider the corresponding decision problem and show that it is NP-complete: given a linear workflow, a mapping and a bound B, does there exist a schedule whose period does not exceed B? This problem is obviously in NP: given a mapping and a schedule, we can check all resource constraints, compute the period K and compare it to B in polynomial time. To establish the completeness, we use a reduction from 2-PARTITION [16]. 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 $\gamma \subset \{1, \dots, n\}$ such that

$$\sum_{i \in \gamma} a_i = \sum_{i \notin \gamma} a_i = B/2 \; ?$$

We associate to $\mathcal{I}_1$ an instance $\mathcal{I}_2$ with $2n+2$ processors, given by the linear workflow and the mapping represented on Figure 4. The corresponding execution graph is given by Figure 5. The size of $\mathcal{I}_2$ is clearly 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.

To give an intuition of the proof, note that processor P_0 has many successors and P_{2n+1} many predecessors (see Figure 5). We need the ordering of the corresponding communications to compute the optimal period for this execution graph.

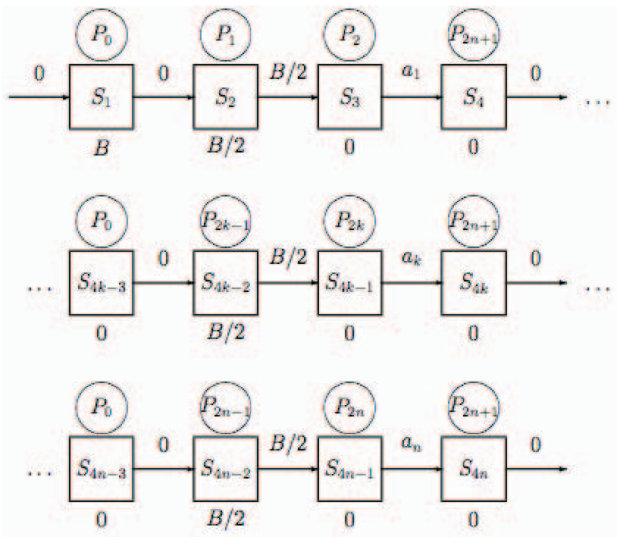

Figure 4: Mapping of the linear workflow

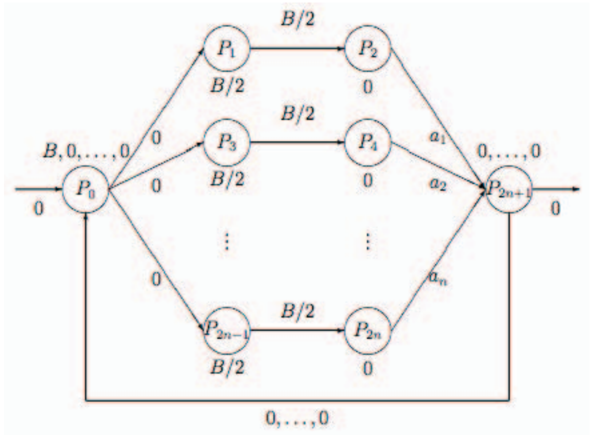

Figure 5: Execution graph for the mapping given in Figure 4.

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 $P_1, P_3, \dots, P_{2n-1}$. All of this is done at the beginning of a period, for different data sets, because these communications have zero cost. Then, P_0 makes its computation which lasts B time units. The schedule of P_0 is represented in Figure 6.

Then, we execute the computations of processors $(P_{2i-1})_{i \in \gamma}$ in the time-interval $[0, B/2]$ and the communications between $(P_{2i-1}, P_{2i})_{i \in \gamma}$ in the time-interval $[B/2, B]$ (see Figure 7). On the contrary, we execute the communications between $(P_{2i-1}, P_{2i})_{i \notin \gamma}$ in the time-interval $[0, B/2]$ and then the computations of processors $(P_{2i-1})_{i \notin \gamma}$ in the time-interval $[B/2, B]$ (see Figure 8). We also execute the communications between $(P_{2i}, P_{2n+1})_{i \in \gamma}$ sequentially and in any order in the time-interval $[0, B/2]$: this is possible because $\sum_{i \in \gamma} a_i = B/2$ by hypothesis (see Figure 9). Similarly, we execute all communications $(P_{2i}, P_{2n+1})_{i \notin \gamma}$ in the time-interval $[B/2, B]$ (see Figure 10). We obtain a schedule of period B, hence a solution to $\mathcal{I}_2$. Note that this construction is possible because $\sum_{i \in \gamma} a_i = \sum_{i \notin \gamma} a_i = B/2$, so we can partition communications to P_{2n+1} so that none starts before $B/2$ and ends after $B/2$ (see Figure 11).

We obtain a schedule of period B, which ends the first part of the proof. One 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 11).

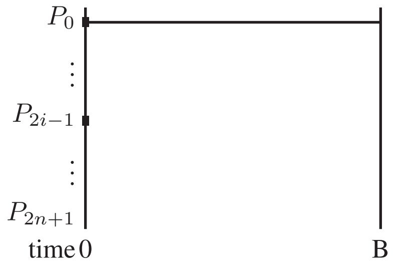

Figure 6: Zero-cost communications and one computation on P_0.

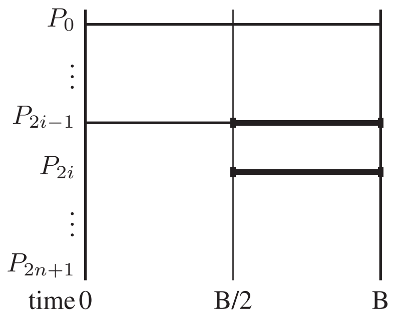

Figure 7: If $i \in \gamma$, there is a computation at $t \in [0, B/2]$ on P_{2i-1}, and a communication at $t \in [B/2, B]$ between P_{2i-1} and P_{2i}.

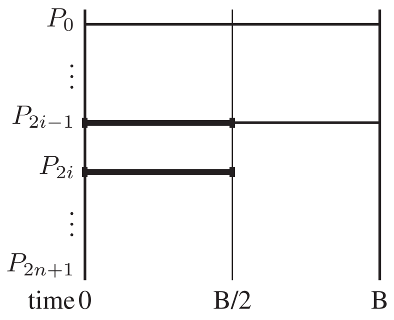

Figure 8: If $i \notin \gamma$, there is a communication at $t \in [0, B/2]$ between P_{2i-1} and P_{2i}, and a computation at $t \in [B/2, B]$ on P_{2i-1}.

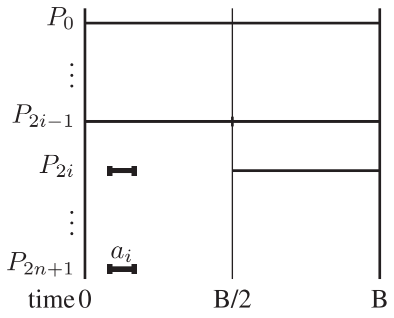

Figure 9: If $i \in \gamma$, the communication between P_{2i} and P_{2n+1} occurs in the time-interval $[0, B/2]$.

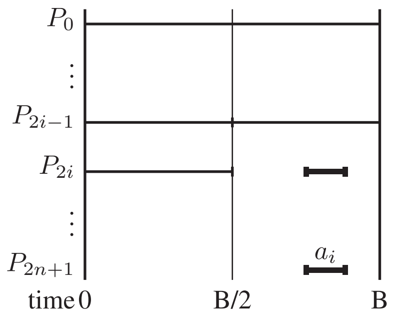

Figure 10: If $i \notin \gamma$, the communication between P_{2i} and P_{2n+1} occurs in the time-interval $[B/2, B]$.

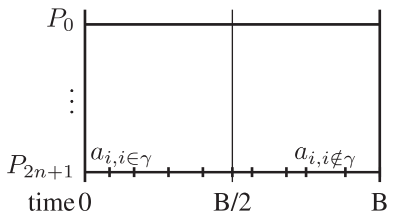

Figure 11: This schedule uses the 2-PARTITION of $\{a_1, \dots, a_n\}$.

Suppose now 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 some 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} at $t \in [t_0, t_0 + B/2]$ and a communication between P_{2i-1} and P_{2i} at $t \in [t_0 + B/2, t_0 + B]$, or a communication between P_{2i-1} and P_{2i} followed by a computation on P_{2i-1}. We define γ as follows:

$$\gamma = \begin{array}{l}\{1 \leqslant i \leqslant n, \text{ there is a computation on } P_{2i-1} \\ \text{at } t \in [t_0, t_0 + B/2]\}.\end{array} \quad (4)$$

Let j be in γ. By hypothesis, there is a computation on P_{2j-1} at $t \in [t_0, t_0 + B/2]$. There is no idle time for P_{2j-1}, so there is a communication between P_{2j-1} and P_{2j} at $t \in [t_0 + B/2, t_0 + B]$. A communication between P_{2j} and P_{2n+1} occurs in the time-interval $t \in [t_0, t_0 + B/2]$. 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 disjoint time-intervals included in $[t_0, t_0 + B/2]$, so we have $\sum_{i\in\gamma} a_i \leqslant B/2$. In the same way, we show that communications between $P_{2i-1}, i \notin \gamma$ and P_{2n+1} are done during disjoint time intervals included in $[t_0 + B/2, t_0 + B]$, hence $\sum_{i\notin\gamma} a_i \leqslant B/2$.

By hypothesis, we have

$$\sum_{i\in\gamma} a_i + \sum_{i\notin\gamma} a_i = \sum_{1\leqslant i\leqslant n} a_i = B \ ,$$

and therefore

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

This concludes the proof. ■

2) *Heuristics and approximation algorithms:*

We now discuss several heuristics to compute a schedule of small period. A first algorithm is to schedule each operation (communication or computation) as soon as possible. This algorithm is called *Greedy Algorithm* (see Algorithm 1). The difficulties that we can face in the Greedy Algorithm come from communications: in the case where there are only computations, each processor can just compute its operations in any order, and the period is the cycle-time of the busiest processor. This leads us to design a new algorithm (see Algorithm 2), which first includes greedily all communications between processors (except P_{in} and P_{out}) in the schedule, and then includes greedily all computations.

Algorithm 1: Greedy Algorithm

```
1 for all operations, communications and computations do
2     add the operation as soon as possible in the schedule
```

Algorithm 2: Communication First Algorithm

```
1 for all communications of the mapping do
2     add the communication as soon as possible in the schedule
3 for all computations of the mapping do
4     add the computation as soon as possible in the schedule
```

Algorithm 3: Longest First Algorithm

```
1 for all operations, communications and computations, from the longest one to the shortest one do
2     add the operation as soon as possible in the schedule
```

In the one-port model, we now exactly how long each operation (communication or computation) will take to execute. 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 $P_{a(k+1)}$ (where $P_{a(k)} \neq P_{a(k+1)}$) lasts $\delta_k / \min(b_{a(k),a(k+1)}, B^o_{a(k)}, B^i_{a(k+1)})$ 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 schedule operations (communications and computations) from the longest to the shortest: if there is a gap in the schedule, a short operation can be more easily inserted than a long one. This leads to Algorithm 3.

We have conducted several tests to experimentally evaluate the previous three algorithms. Because results may depend heavily on application and platform parameters, we did experiments on a large number of random graphs, generated with both uniform and Gaussian distributions. See Figure 12 for sample results (please refer to the companion research report [24] for extended data).

Uniform distributions.

First we varied the number n of stages in the application. We generated different sets of graphs $\{set_1, \dots, set_k\}$, where the number of stages in set_i is $n_i = 10 \times i$ for $i \in [\![1, 10]\!]$. To generate a graph of the set set_i, we have to decide for the values of $\{\delta_0, \dots, \delta_{n_i}\}$ and $\{c_1, \dots, c_{n_i}\}$. A first method is to choose four integers (a, b, c, d) such that $a \leqslant b$ and $c \leqslant d$ and to pick any δ_j randomly, uniformly in $[\![a, b]\!]$ and any c_l randomly, uniformly in $[\![c, d]\!]$. In our experiments, typical values for $[\![a, b]\!]$ are $[\![0, 9]\!]$, $[\![0, 99]\!]$ and $[\![0, 999]\!]$ and typical values of $[\![c, d]\!]$ are $[\![0, 0]\!]$, $[\![0, 9]\!]$, $[\![0, 99]\!]$ and $[\![0, 999]\!]$. As for platform parameters, because increasing the speed of a processor or the bandwidth of a link is similar to decreasing the value of some computation or of some communication, we can safely consider that processor speeds and bandwidths are all equal to 1. We used such homogeneous platforms with p processors, where values of p are typically 3, 10 or 20. Because computing the optimal period $K_{\min}$ is difficult (Theorem 2), we compared the period returned by the three algorithms with the cycle-time CT, which is an easy-to-compute lower bound of $K_{\min}$.

A first result is that periods found by the Greedy Algorithm are, in average, larger than periods found by the Communication First Algorithm, which, in turn, are larger than those obtained by the Longest First Algorithm. More importantly, all algorithms appear to construct schedules of small periods, in comparison with the cycle-time: if K_{GA}, K_{CFA} and K_{LFA} are periods found respectively by the Greedy Algorithm, the Communication First Algorithm and the Longest First Algorithm, we have, in average, $K_{GA} = 1.036CT$, $K_{CFA} = 1.028CT$ and $K_{LFA} = 1.012CT$, for $(a, b, c, d) = (0, 999, 0, 999)$ and on $p = 10$ processors. This might lead to conjecture that these algorithms are good approximations of the cycle-time, and consequently of the optimal period.

Gaussian distributions.

Random graphs with uniform distributions are not representative of all application graphs. In particular, there is very little chance that a random graph has one huge computation, and many small other computations and communications. Another way to generate random graphs is to pick any δ_i and c_i using a Gaussian distribution: δ_i or c_i is given by the absolute value of a random variable following a centered normal law $\mathcal{N}(0, 1000)$. Results obtained for these graphs are similar to previous ones, and we have $K_{GA} = 1.037CT$, $K_{CFA} = 1.025CT$ and $K_{LFA} = 1.010CT$ on $p = 10$ processors.

Altogether, all algorithms appear to be good approximations of the cycle-time CT, and consequently of the optimal period $K_{\min}$. In all tests (both uniform and Gaussian), returned periods never exceeded twice the cycle-time, and it seemed reasonable to conjecture that all algorithms are some $2-$approximations. Despite these promising experimental results, it turns out that only the Longest First Algorithm is an approximation algorithm, and it is not a $2-$approximation but a $4-$approximation.

Proposition 3. *There is no constant k such that the Greedy Algorithm (resp. the Communication First Algorithm) computes a schedule whose period is a $k-$approximation of the optimal period.*

The proof of the above proposition involves a complicated counter-example where task sizes are exponential in the number of tasks, see [24].

Theorem 4. *The Longest First Algorithm computes a schedule whose period is a $4-$approximation of the cycle-time $CT(a, G)$ (hence of the optimal period).*

Proof: Recall that the Longest First Algorithm adds tasks (communications and computations) *from the longest to the shortest*, and as soon as possible, in the schedule. Hence assume that tasks are given by the ordered list $(t_i)_{1 \leqslant i \leqslant N}$, where t_1 is the longest task and t_N the shortest one. The size of task t_i is given by size(t_i). We proceed by induction on the number of tasks inserted into the schedule.

Assume that the schedule contains the first k tasks $t_1, \dots, t_k$. The current completion time of P_i, $1 \leqslant i \leqslant p$, is denoted as

Size of linear graph	10	20	30	40	50	60	70	80	90	100
Number of tests	1000	950	900	850	800	750	700	650	600	550
Greedy	3554.76	5217.48	6559.74	7953.39	9095.66	10325.8	11453.3	12580.3	13739.4	14719.2
Communication First	3426.55	5091.81	6425.09	7826.91	8960.36	10212.6	11335	12447	13615.2	14615.7
Longest First	3371.81	4997.29	6303.7	7668.32	8776.25	10011.7	11127.5	12208.3	13366.6	14376.3
Cycle-time	3290.37	4890.45	6183.69	7548.35	8638.13	9878.1	10976.5	12049.1	13220.5	14207.6

(a) Uniform distributions: average results with $p = 20$, $(a, b) = (c, d) = (0, 999)$.

Size of linear graph	10	20	30	40	50	60	70	80	90	100
Number of tests	1000	950	900	850	800	750	700	650	600	550
Greedy	6919.04	10771.3	14335	17789.5	20629.7	23527.1	26691.9	29412.7	32525.5	35320.3
Communication First	6680.3	10481.7	14006.9	17455	20313.3	23182.6	26294.8	29029	32179.5	34919.2
Longest First	6597.3	10305.3	13769.3	17162.4	20003.8	22841.7	25885.6	28584.1	31692.1	34396.3
Cycle-time	6432.42	10095.3	13542.7	16925.2	19728.3	22538.4	25576.5	28267.5	31382.7	34067.2

(b) Gaussian distributions: average results with $p = 10$ and the normal law $\mathcal{N}(0, 1000)$.

Figure 12: Sample simulation results for the three algorithms.

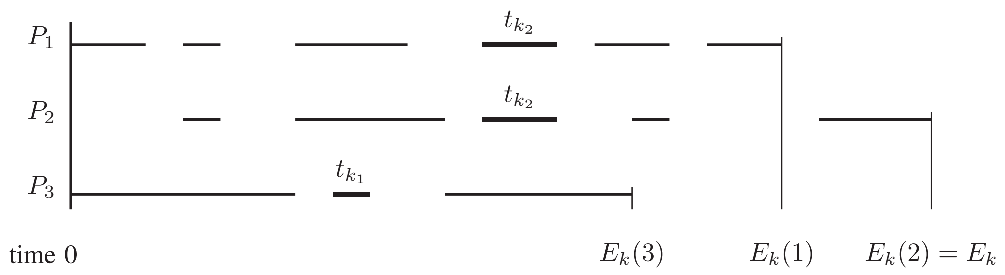

Figure 13: With 3 processors P_1, P_2 and P_3, and k tasks in the schedule. Two tasks are represented in bold: a *computation* t_{k_1} and a *communication* t_{k_2}.

$E_k(i)$, and let $E_k = \max_{1 \leqslant i \leqslant p} \{E_k(i)\}$. These notations are illustrated on Figure 13.

Let $CT = CT(a, G)$. We prove by induction that:

$$\forall\ 0 \leqslant k \leqslant N,\ E_k \leqslant 4 \times CT. \tag{5}$$

The base case is trivial since $E_0 = 0$. Suppose that the algorithm has constructed a schedule with the first k tasks, such that Equation (5) is verified. When the Longest First Algorithm adds the task t_{k+1} of size size(t_{k+1}), there are two possibilities: either E_k is modified, or not. Obviously, if E_k is not modified, by hypothesis $E_{k+1} = E_k \leqslant 4 \times CT$, and the property holds for $k + 1$ tasks.

Now, if $E_{k+1} \neq E_k$, we have $E_{k+1} > E_k$, because adding a task t_{k+1} can only increase the completion time of a processor. Hence there exist two indices $i_1, i_2 \in \{1, \ldots, p\}^2$ such that

$$\begin{cases} E_{k+1}(i_2) > E_k(i_1) \geqslant E_k(i_2); \\ \max_{1 \leqslant l \leqslant p}\{E_k(l)\} = E_k(i_1); \\ \max_{1 \leqslant l \leqslant p}\{E_{k+1}(l)\} = E_{k+1}(i_2). \end{cases}$$

There exists a processor P_{i_1} such that $E_k = E_k(i_1)$. When we add the new task t_{k+1} on processor P_{i_2}, the algorithm puts this task at the end of its schedule. The completion time of processor P_{i_2} is increased from $E_k(i_2)$ to $E_{k+1}(i_2) \geqslant E_k(i_2) + \text{size}(t_{k+1})$ (this is not necessarily an equality because the algorithm may leave a gap on processor P_{i_2} before putting task t_{k+1}).

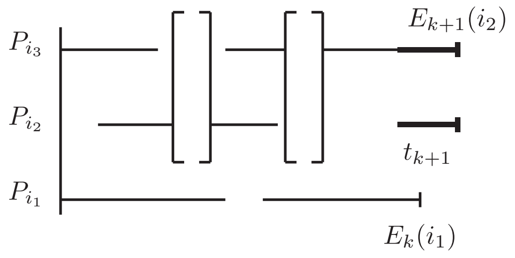

Figure 14: Adding a *communication* in a schedule using the Longest First Algorithm. Common gaps between processors P_{i_2} and P_{i_3} are represented between brackets.

Suppose first that 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 14. 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 task t_{k+1} in the schedule (represented between brackets in Figure 14) is bounded by the number of tasks on P_{i_2} plus the number of tasks on P_{i_3} at stage $k+1$:

$$\begin{aligned}\#_{\text{common gaps}}(P_{i_2}, P_{i_3}, k+1) \leqslant \#(l \leqslant k+1, t_l \text{ is on } P_{i_2}) \\ +\#(l \leqslant k+1, t_l \text{ is on } P_{i_3}).\end{aligned} \tag{6}$$

Moreover, there is no common gap in the schedule of size larger than size(t_{k+1}), because the Longest First Algorithm would have put the task t_{k+1} inside this interval. 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 $k+1$ is bounded by the number of common gaps times the biggest size of a common gap:

$$\begin{aligned}S(P_{i_2}, P_{i_3}, k+1) \quad \leqslant \quad & \text{size}(t_{k+1})\times \\ & \#_{\text{common gaps}}(P_{i_2}, P_{i_3}, k+1).\end{aligned} \tag{7}$$

Combining (6) and (7) leads to

$$\begin{aligned}& S(P_{i_2}, P_{i_3}, k+1) \\ & \quad \leqslant \text{size}(t_{k+1}) \times \# (l \leqslant k+1, t_l \text{ is on } P_{i_2}) \\ & \qquad +\text{size}(t_{k+1}) \times \# (l \leqslant k+1, t_l \text{ is on } P_{i_3}) .\end{aligned} \tag{8}$$

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

$$\begin{aligned}& \text{size}(t_{k+1}) \times \# (l \leqslant k, t_l \text{ is on } P_{i_2}) \\ & \quad \leqslant \textstyle\sum_{l=1}^{k+1} \text{size}(t_l) \mathbb{1}_{t_l \text{ is on } P_{i_2}} \leqslant CT_{i_2};\end{aligned}$$

$$\begin{aligned}& \text{size}(t_{k+1}) \times \# (l \leqslant k, t_l \text{ is on } P_{i_3}) \\ & \quad \leqslant \textstyle\sum_{l=1}^{k+1} \text{size}(t_l) \mathbb{1}_{t_l \text{ is on } P_{i_3}} \leqslant CT_{i_3}.\end{aligned}$$

Combining these equations with (8), we derive

$$\begin{aligned}S(P_{i_2}, P_{i_3}, k+1) \quad & \leqslant \quad CT_{i_2} + CT_{i_3} \\ & \leqslant \quad 2 \max_{1 \leqslant i \leqslant p} \{CT_i\} = 2 \times CT.\end{aligned} \tag{9}$$

The value of $E_{k+1}(i_2)$ (see Figure 14) is smaller than the size of the common gaps between P_{i_2} and P_{i_3} at stage $k+1$ 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 as:

$$E_{k+1}(i_2) \leqslant S(P_{i_2}, P_{i_3}, k+1) + CT_{i_2} + CT_{i_3}. \tag{10}$$

Combining this equation with (9), we obtain

$$\begin{aligned}E_{k+1} = E_{k+1}(i_2) \quad & \leqslant \quad 2 \times CT + CT_{i_2} + CT_{i_3} \\ & \leqslant \quad 4 \times CT,\end{aligned} \tag{11}$$

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

Suppose now that t_{k+1} is a computation on processor P_{i_2}. Then 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 reasoning. This ends the induction step when t_{k+1} is a computation. Letting $k = N$ in Equation (5), we conclude that

$$E_N \leqslant 4 \times CT,$$

which concludes the proof. ■

Theorem 5. *For any $\beta < 4$, there exists a linear workflow and a mapping such that the Longest First Algorithm computes a schedule of period $K > \beta K_{\min}$: the Longest First Algorithm is not a $k-$approximation for any k smaller than 4.*

As in the proof of Proposition 3, the proof of this theorem consists in exhibiting a counter-example. The graph is built by induction so as to maximize the size of the gaps, and the construction is very sophisticated. Indeed, in most practical situations, the algorithm always is a $2-$approximation, see the simulation results in Figure 12. The detailed proofs of these results are available in [24].

C. Bi-criteria

Finally, because the period minimization problem is NP-hard, the bi-criteria problem that enforces both a prescribed period and latency is NP-hard too.

IV. Multi-port model

In this section, we explore the problem of period and/or latency minimization in the multi-port model.

A. Latency

Theorem 6. *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.*

Proof: The proof is the same as for the one-port model. In fact, the optimal latency has the same value for both models. ■

B. Period

Theorem 7. *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, $K_{\min} = CT(a, G)$, where $CT(a, G)$ is the cycle-time.*

Proof: In a nutshell, the construction of the schedule is such that all input communications and all output communications are done in parallel, each of them using only a fraction of the bandwidth. Computations are done sequentially.

Formally, let $CT = CT(a, G)$, we already know that $CT \leqslant K_{\min}$. We compute in polynomial time a schedule which respects all the constraints and that has a period $K = CT$.

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

$$K \geqslant \max_{u \in \{1,\dots,p\}} \left\{ \frac{1}{s_u} \sum_{a(i)=u} w_i \right\}. \tag{12}$$

Similarly, we can bound the period by the minimum amount of time-units needed for the communications between any two processors P_u and P_v with $u \neq v$, or for the input/output communications of every network card. Combining all these bounds leads to $K \geqslant Q$, where

$$Q = \max \begin{cases} \max\limits_{u \in \{1,\dots,p\}} \left\{ \frac{1}{s_u} \sum\limits_{a(i)=u} w_i \right\} \\ \max\limits_{u,v \in \{1,\dots,p\} \cup \{in,out\}, u \neq v} \left\{ \frac{1}{b_{u,v}} \sum\limits_{a(i)=u, a(i+1)=v} \delta_i \right\} \\ \max\limits_{u \in \{1,\dots,p\} \cup \{in\}} \left\{ \frac{1}{B_u^o} \sum\limits_{a(i)=u} \delta_i \right\} \\ \max\limits_{u \in \{1,\dots,p\} \cup \{out\}} \left\{ \frac{1}{B_u^i} \sum\limits_{a(i)=u} \delta_{i-1} \right\} \end{cases} \quad (13)$$

We check that $Q = CT$, by definition of the cycle-time CT_u of P_u (see Equation (2)) and by definition of CT (see Equation (3)). We now give a schedule whose period is $K = CT$ and which is computable in polynomial time. Assume that the k-th data set is input at step $t_0 + k.CT$. For each data set, any processor P_u has three kinds of tasks to execute: computations for stages $\{S_i\}_{a(i)=u}$, input communications between stages $\{S_{i-1}, S_i\}_{a(i)=u}$ and output communications between stages $\{S_i, S_{i+1}\}_{a(i)=u}$. We describe how these tasks can all be computed in parallel within a time-interval $[t_0, t_0 + CT]$. By periodicity, this describes the whole schedule.

Input communications are performed in parallel. They begin at time t_0, end at time t_0+CT, and any communication between stages S_{i-1} and S_i uses a bandwidth fraction equal to δ_{i-1}/CT. Similarly, output communications are done in parallel, and any communication between S_i and S_{i+1} uses a bandwidth fraction equal to 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 δ_i/CT to a processor P_{u_2}, then processor P_{u_2} receives a data of size δ_i from P_{u_1} using the same bandwidth. Moreover, the total 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

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

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 execute the computations $\{w_i\}_{a(i)=u}$ between t_0 and $t_0 + CT$, and concludes the proof.

■

C. Bi-criteria

Theorem 8. *Given a linear workflow and a mapping in the multi-port model:*

- *the problem of computing the schedule that minimizes the period at minimal latency $L_{\min}$ has polynomial complexity;*
- *the problem of computing a schedule that minimizes the latency at minimal period $K_{\min}$ is NP-hard.*

Proof: The whole proof is available in [24].

We outline here the NP-completeness reduction for the second part of the theorem, because it is rather involved: given a linear graph, a mapping, and a bound L, does there exist a schedule whose period is the minimal period $K_{\min}$ and whose latency does not exceed L?

We use a reduction from 2-PARTITION [16] and consider an instance $\mathcal{I}_1$ with n integers a_i, $1 \leqslant i \leqslant n$ such that $\sum_{i=1}^n a_i = K$: does there exist $\gamma \subset \{1, \dots, n\}$ such that

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

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 15. The platform is fully homogeneous and the common computation speed is $s = 1$. There are no communication costs. The minimal period of the mapping is $K_{\min} = K$.

We show that $\mathcal{I}_1$ has a solution if and only if we can design a schedule for $\mathcal{I}_2$ whose latency is $L = (n + 2)K$. We construct such a schedule of period K as represented on Figure 16, where 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 S_{2n+4} all 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} does not have to wait before being executed. ■

As a consequence of Theorem 8, the bi-criteria problem (computing a schedule that respects both a period K and a latency L) is NP-hard with the multi-port model.

V. Conclusion

This work presents complexity results for finding optimal schedules for linear workflows, when the mapping is given, both for the one-port model (without overlap) and for the multi-port 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 objective functions and two models, we present six main complexity results. In both models, latency minimization is easy; period minimization can be done in polynomial time in the multi-port model while it is NP-hard in the one-port model. For both models, the bi-criteria problem is NP-hard. We also

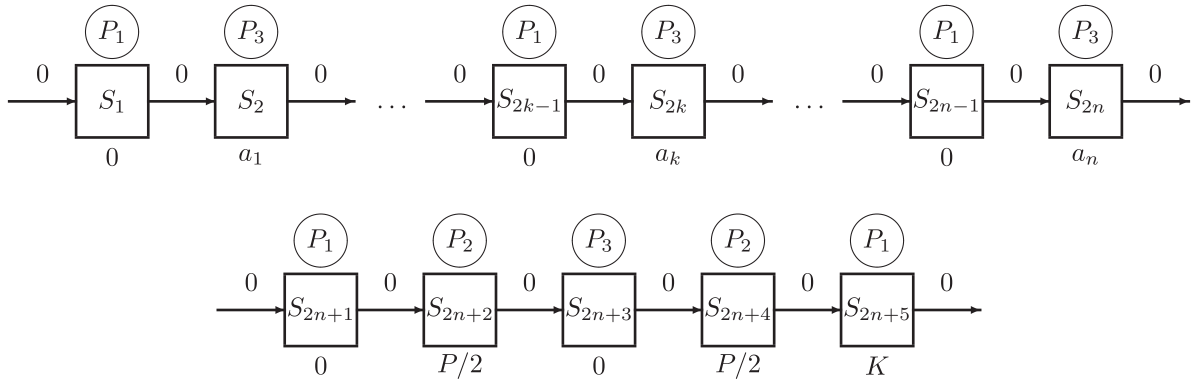

Figure 15: Representation of instance $\mathcal{I}_2$.

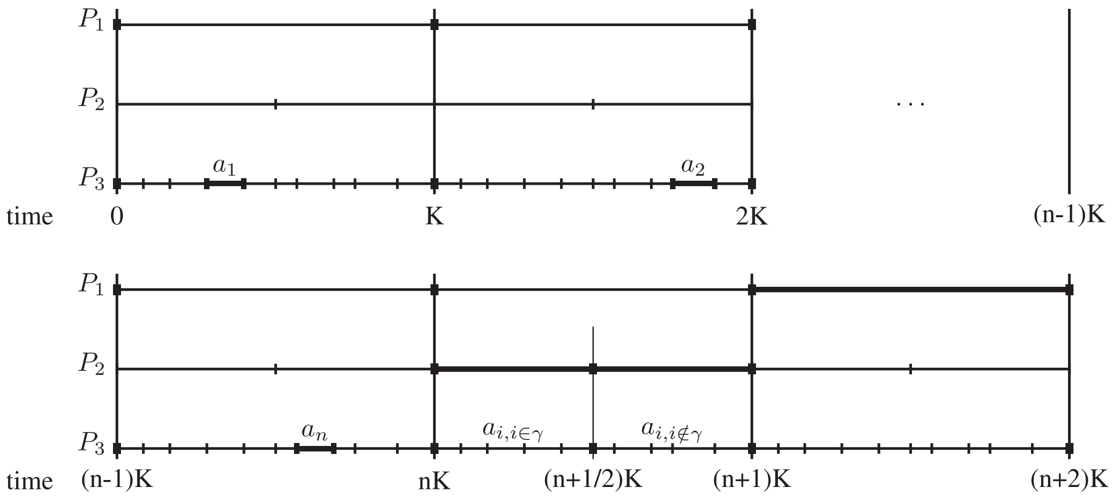

Figure 16: Representation of a schedule of period K, for the mapping given by Figure 15. The computation of one data set is represented in bold. The latency is $(n+2)K = L$.

provide a $4-$approximation algorithm for period minimization in the one-port model.

In the future, we plan to find approximation algorithms, or at least efficient heuristics, for the bi-criteria problem for both models. This is an important question since real world problems often have constraints on both latency and period. In particular, many problems, especially in the domain of real time systems, require period maximization while obeying strict latency constraints.

We would also like to extend our models to include preemption. In both one-port and multi-port models, we assume that once a task starts, it runs until completion. In many modern processors, this might be a pessimistic assumption. We may be able to suspend an already running task, run a higher priority task, and then resume the prior task in some systems. With unlimited preemption, it appears that minimizing the period will also take polynomial time. However, even on machines that allow preemption, there is usually a cost to interrupting a task. Therefore, unlimited preemption is an unrealistic model. In the future, we would like to work on designing reasonable models that include preemption, but carefully consider the cost of interruptions. We then want to assess both the mapping and the scheduling problems for linear workflows for these preemptive models.

Another avenue for future work is to consider the scheduling problem (given the mapping) for general DAGs. Of course, our NP-completeness results generalize. However, we saw in this paper that latency minimization is easy for linear workflows. Latency minimization may not remain so easy for general DAGs. In addition, period minimization for the multi-port model is a polynomial time problem for linear workflows, but may be considerably harder for general DAGs. Even for problems that are NP-complete, it may be possible to find approximation algorithms or good heuristics for period or latency minimization of general DAGs.

Acknowledgment: A. Benoit and Y. Robert are with the Institut Universitaire de France, and their work was supported in part by the ANR StochaGrid project. We would like to thank Fanny Dufossé for her help in the proof of Theorem 2.

References

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

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

[3] S. L. Hary and F. Ozguner, "Precedence-constrained task allocation onto point-to-point networks for pipelined execution," *IEEE Trans. Parallel and Distributed Systems*, vol. 10, no. 8, pp. 838–851, 1999.

[4] Q. Wu, J. Gao, M. Zhu, N. Rao, J. Huang, and S. Iyengar, "On optimal resource utilization for distributed remote visualization," *IEEE Trans. Computers*, vol. 57, no. 1, pp. 55–68, 2008.

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

[6] 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*. Springer, 2003, pp. 1–17.

[7] J. Gummaraju, J. Coburn, Y. Turner, and M. Rosenblum, "Streamware: programming general-purpose multicore processors using streams," in *13th Int. Conf. ASPLOS'2008.* ACM Press, 2008, pp. 297–307.

[8] R. Stephens, "A survey of stream processing," *Acta Informatica*, vol. 34, no. 7, pp. 491–541, 1997.

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

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

[11] ——, "A duplication based algorithm for optimizing latency under throughput constraints for streaming workflows," in *ICPP'2008, the Int. Conf. on Parallel Processing*. IEEE Computer Society Press, 2008, pp. 254–261.

[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*. ACM Press, 1995, pp. 134–143.

[13] ——, "Optimal latency-throughput tradeoffs for data parallel pipelines," in *ACM Symposium on Parallel Algorithms and Architectures SPAA'96*. ACM Press, 1996, pp. 62–71.

[14] A. Benoit and Y. Robert, "Mapping pipeline skeletons onto heterogeneous platforms," *J. Parallel Distributed Computing*, vol. 68, no. 6, pp. 790–808, 2008.

[15] 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," LIP, ENS Lyon, France, Research Report RR-2004-20, Apr. 2004, available at the url http://graal.ens-lyon.fr/~yrobert/.

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

[17] K. Agrawal, A. Benoit, and Y. Robert, "Mapping linear workflows with computation/communication overlap," in *ICPADS'2008, the 14th IEEE Int. Conf. on Parallel and Distributed Systems*. IEEE CS Press, 2008, pp. 195–202.

[18] K. Agrawal, A. Benoit, F. Dufossé, and Y. Robert, "Mapping filtering streaming applications with communication costs," in *21st ACM Symposium on Parallelism in Algorithms and Architectures SPAA 2009*. ACM Press, 2009.

[19] T. Yang and A. Gerasoulis, "DSC: Scheduling parallel tasks on an unbounded number of processors," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 9, pp. 951–967, 1994.

[20] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys*, vol. 31, no. 4, pp. 406–471, 1999.

[21] H. Topcuoglu, S. Hariri, and M. Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[22] P. Bhat, C. Raghavendra, and V. Prasanna, "Efficient collective communication in distributed heterogeneous systems," in *ICDCS'99 19th Int. Conf. on Distributed Computing Systems*. IEEE Computer Society Press, 1999, pp. 15–24.

[23] B. Hong and V. Prasanna, "Bandwidth-aware resource allocation for heterogeneous computing systems to maximize throughput," in *Proc. 32th Int. Conf. on Parallel Processing (ICPP'2003)*. IEEE Computer Society Press, 2003.

[24] K. Agrawal, A. Benoit, L. Magnan, and Y. Robert, "Scheduling algorithms for workflow optimization," LIP, ENS Lyon, France, Research Report 2009-22, Jul. 2009, available at http://graal.ens-lyon.fr/~yrobert/.