Mapping linear workflows with computation/communication overlap

Kunal Agrawal[1], Anne Benoit[2] and Yves Robert[2]

[1] CSAIL, Massachusetts Institute of Technology, USA
[2] LIP, École Normale Supérieure de Lyon, France
kunal_ag@mit.edu,{Anne.Benoit|Yves.Robert}@ens-lyon.fr

June 2008

LIP Research Report RR-2008-21

Abstract

This paper presents theoretical results related to mapping and scheduling linear workflows onto heterogeneous platforms. We use a realistic architectural model with bounded communication capabilities and full computation/communication overlap. This model is representative of current multi-threaded systems. In these workflow applications, the goal is often to maximize throughput or to minimize latency. We present several complexity results related to both these criteria.

To be precise, we prove that maximizing the throughput is NP-complete even for homogeneous platforms and minimizing the latency is NP-complete for heterogeneous platforms. Moreover, we present an approximation algorithm for throughput maximization for linear chain applications on homogeneous platforms, and an approximation algorithm for latency minimization for linear chain applications on all platforms where communication is homogeneous (the processor speeds can differ). In addition, we present algorithms for several important special cases for linear chain applications. Finally, we consider the implications of adding feedback loops to linear chain applications.

Key words: pipeline graphs, workflow, scheduling, mapping, period, latency, feedback loop, complexity results.

1 Introduction

Pipelined workflows are a popular programming paradigm for streaming applications like video and audio encoding and decoding, DSP applications etc. Streaming applications are becoming increasingly prevalent, and many languages [27, 19, 8, 12] are being continually designed to support these applications (see [23] for survey of older streaming languages). In these languages, the programmer expresses its programs by creating a ***workflow graph***, and the system maps this workflow graph on a target machine.

A workflow graph contains several ***stages***, and these stages are connected to each other using first-in-first-out ***channels***. Data is input into the graph using input channel(s) and the outputs are produced on the output channel(s). Since data continually flows through these applications, the goal of a scheduler is often to increase the ***throughput*** and/or decrease the ***latency***. One can think of these applications as a graph, where stages are the nodes and the channels are the edges. Most of the problems related to mapping general graphs optimally are rather obviously NP-Complete; therefore, we consider the special case of linear chains where they may or may not have feedback edges. In this domain, we provide comprehensive theoretical analysis of the complexity of mapping these graphs on both homogeneous and heterogeneous distributed memory platforms.

Subhlok and Vondran [24, 25] studied the problem of mapping linear chain graphs on homogeneous platforms, and these complexity results were extended for heterogeneous platforms under the ***one-port model*** in [3]. In this model, the processor can either compute, receive an incoming communication, or send an outgoing communication at a time. This model does a good job of representing single-threaded systems. Unfortunately, this model is not suitable for handling general mappings, or applications which have feedback, and it can deadlock in this case. Therefore, in this paper, we explore the ***bounded-multiport model***, which allows multiple incoming and outgoing communications simultaneously, and allows the computation and communication to ***overlap***. To the best of our knowledge, the bounded-multiport model with overlap has not been explored for linear chains, and we start to explore its complexity before extending the approach to applications with feedback.

We consider three kinds of platforms for mapping these applications. *Fully Homogeneous* platforms are those which have identical processors. That is, all processors run at the same speed, and communicate with each other using links of the same bandwidth. *Communication Homogeneous* platforms are those in which the processors may have different speeds, but they are all connected by identical communication interconnect. *Fully Heterogeneous* platforms are those where both processor speeds and the speed of the interconnect changes from processor to processor.

Here is the summary of our results:

- Finding the mapping with optimal throughput is NP-complete even for *Fully Homogeneous* platforms in the bounded multiport model with overlap. Note that this result implies that it is NP-complete for *Communication Homogeneous* and *Fully Heterogeneous* platforms as well. Finding the mapping with optimal latency is NP-complete for *Communication Homogeneous* platforms. The problem of finding the mapping with optimal latency for *Fully Homogeneous* platforms is left open. These proofs are given in Section 4.

- Section 5 shows finding the best ***interval mappings*** for both throughput and latency can be done in polynomial time for *Fully Homogeneous* platforms. These interval mappings are commonly used since they minimize the communication overhead. However, finding optimal ***interval mappings*** is NP-complete for *Communication Homogeneous* platforms.

- Since finding the best mapping is NP-complete, we looked for approximation algorithms. In Section 6, we show that interval mappings provide a 2-approximation for throughput on *Fully Homogeneous* platforms. Since these interval mappings can be found in polynomial time, we have a 2-approximation algorithm for the throughput. Unfortunately, this approximation result no longer holds for *Communication Homogeneous* platform. Mapping all the stages on the same processor provides a 1.5-approximation for latency on *Communication Homogeneous* platforms (hence also for *Fully Homogeneous* platforms). However, this result does not hold for *Fully Heterogeneous* platforms.

- In the special case where an infinite number of processors are available to the application, interval mappings are optimal for throughput. Similarly, interval mappings are optimal for throughput if the application is regular — all the stages perform the same amount of computation and communication. Therefore, for both these special cases, we have polynomial time algorithms for linear chains on *Fully Homogeneous* platforms for throughput. These results are explained in Section 7.

- In Section 8, we show that almost all problems become more difficult when feedback loops are added. Interval mappings are no longer a good approximation for throughput even for *Fully Homogeneous* platforms. However, mapping all stages on the same processor is still a good approximation for latency for *Communication Homogeneous* platforms. In addition, we prove that optimizing throughput is NP-complete even for infinite number of processors when we have feedback loops.

2 Related Work

This section describes some of the related work that has not been mentioned in Section 1.

An important and representative project in the field of scheduling workflows is the DataCutter [9] project. One goal of this project is to schedule multiple data analysis operations onto clusters and grids, and to decide where to place and/or replicate various components [4, 5, 21]. A typical application is a chain of consecutive filtering operations, to be executed on a very large data set. The task graphs targeted by DataCutter are usually tree-shaped, which makes it easier to design efficient heuristics to solve the previous placement and replication optimization problems. However, recent papers [29, 30] target workflows structured as arbitrary DAGs (Directed Acyclic Graphs) and consider bi-criteria optimization problems on homogeneous platforms. These papers provide many interesting ideas and several heuristics to solve the general mapping problem.

In [26], Taura and Chien consider applications composed of several copies of the same task graph, expressed as a DAG. These copies are to be executed in pipeline fashion. Their problem is shown NP-complete, and they provide an iterative heuristic to determine a good mapping. At each step, the heuristic refines the current clustering of the DAG. Beaumont et al [2] consider the same problem as Taura and Chien, i.e., with a general DAG, but they allow a given task type to be mapped onto several processors, each executing a fraction of the total number of tasks. The problem remains NP-complete, but becomes polynomial for special classes of DAGs, such as series-parallel graphs. For such graphs, it is possible to determine the optimal mapping owing to an approach based upon a linear programming formulation.

Since most problems for general DAG applications are shown to be NP-hard, we focus in this paper on a limited class of application graphs (linear chains, with and without feedback).

3 Framework

In this section we first describe the application model and architectural framework. Then we detail mapping rules and objectives.

3.1 Application Model

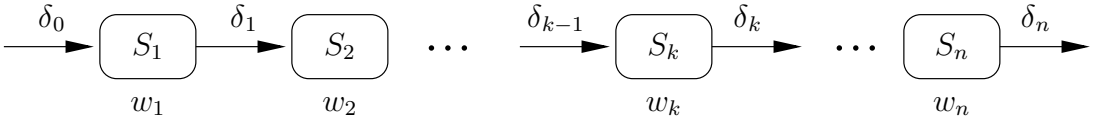


Figure 1: Application linear chain.

We consider simple application workflows whose graphs are a pipeline (i.e., a linear chain). Such graphs are representative of a wide class of applications, and constitute the typical building blocks upon which to build and execute more complex workflows. A pipeline graph of n stages S_k, $1 \leq k \leq n$ is illustrated on Figure 1. Consecutive data sets are fed into the pipeline and processed from stage to stage, until they exit the pipeline after the last stage.

Each stage executes a task. More precisely, the k-th stage S_k receives an input from the previous stage, of size δ_{k-1}, performs a number of w_k computations, and outputs data of size δ_k to the next stage. This operation corresponds to the k-th task and is repeated periodically on each data set. Communications and computations are done in parallel, thus input for data set $i+1$ is received while computing for data set i and sending result for data set $i-1$. The first stage S_1 receives an input of size δ_0 from the outside world, while the last stage S_n returns the result, of size δ_n, to the outside world.

3.2 Execution Model

3.2.1 Platform graph

We target a heterogeneous platform with p processors P_u, $1 \leq u \leq p$, fully interconnected as a (virtual) clique. There is a bidirectional link $link_{u,v} : P_u \rightarrow P_v$ between any processor pair P_u and P_v, of bandwidth $b_{u,v}$. Note that we do not need to have a physical link between any processor pair. Instead, we may have a switch, or even a path composed of several physical links, to interconnect P_u and P_v; in the latter case we would retain the bandwidth of the slowest link in the path for the value of $b_{u,v}$. In the most general case, we have fully heterogeneous platforms, with different processors speeds and link capacities. The speed of processor P_u is denoted as s_u, and it takes X/s_u time-units for P_u to execute X floating point operations. We also enforce a linear cost model for communications, hence it takes $X/b_{u,v}$ time-units to send (resp. receive) a message of size X to (resp. from) P_v.

Finally, there are additional constraints on the communications: in addition to link bandwidths, we deal with processor network cards and we bound the total communication capacity of each computing resource: we denote by B_u^i (resp. B_u^o) the capacity of the input (resp. output) network card of processor P_u. In other words, P_u cannot receive more than $1/\mathsf{B}_u^i$ data items per time-unit, and it cannot send more than $1/\mathsf{B}_u^o$ data items per time-unit.

We classify below particular cases which are important, both from a theoretical and practical perspective:

Fully Homogeneous– These platforms have identical processors ($s_u = s$) and homogeneous communication devices ($b_{u,v} = b$ for link bandwidths, and $\mathsf{B}^i_u = \mathsf{B}^i$, $\mathsf{B}^o_u = \mathsf{B}^o$ for network cards). They represent typical parallel machines.

Communication Homogeneous– These platforms have different-speed processors ($s_u \neq s_v$) but are interconnected with homogeneous communication devices: links and network cards of same capacities ($b_{u,v} = b$, $\mathsf{B}^i_u = \mathsf{B}^i$, $\mathsf{B}^o_u = \mathsf{B}^o$). They correspond to networks of workstations with plain TCP/IP interconnects or other LANs.

Fully Heterogeneous– These are the most general, fully heterogeneous architectures, with $s_u \neq s_v$, $b_{u,v} \neq b_{u',v'}$, $\mathsf{B}^i_u \neq \mathsf{B}^i_v$, $\mathsf{B}^o_u \neq \mathsf{B}^o_v$. Hierarchical platforms made up with several clusters interconnected by slower backbone links can be modeled this way.

Finally, we assume that two special additional processors P_{in} and P_{out} are devoted to input/output data. Initially, the input data for each task resides on P_{in}, while all results must be returned to and stored in P_{out}. Of course we may have a single processor acting as the interface for the computations, i.e., $P_{in} = P_{out}$.

3.2.2 Realistic Communication Models

The standard model for DAG scheduling heuristics [31, 17, 28] 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 [6, 7]. In this model, a given processor can be involved in a single communication at any time-step, either a send or a receive. However, independent communications between distinct processor pairs can take place simultaneously. The one-port model seems to fit the performance of some current MPI implementations, which serialize asynchronous MPI sends as soon as message sizes exceed a few megabytes [20].

The one-port model fully accounts for the heterogeneity of the platform, as each link has a different bandwidth. It generalizes simpler models [1, 18, 15] where communication time only depends on the sender, not on the receiver. In these models, the communication speed from a processor to all its neighbors is the same. A study of mapping strategies for linear chain application graphs under the one-port model has been conducted in [3].

Another realistic model is the *bounded multiport* model [13]. In this model, the total communication volume outgoing from a given node is bounded (by the capacity of its network card), but several communications along different links can take place simultaneously (provided that the link bandwidths are not exceeded either). We point out that recent multi-threaded communication libraries such as MPICH2 [11, 14] now allow for initiating multiple concurrent send and receive operations, thereby providing practical realizations of the multiport model.

3.2.3 Computation/Communication Overlap

Another key assumption to define the execution model is to decide whether computation can overlap with (independent) communication. Most state-of-the-art processors running a threaded

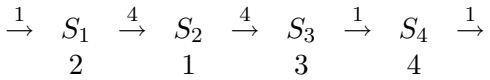


Figure 2: Toy example to explain how period and latency are determined.

operating system are indeed capable of such an overlap (even though it may be at the price of some degradation in the computing speed [16]).

The main emphasis of this paper is to investigate the complexity of various mapping problems under the bounded multiport model with computation/communication overlap. These two assumptions (multiport and overlap) fit well together because they both require a multi-threaded system. However, they turn out to have a tremendous impact on the definition of the throughput and of the latency that can be achieved: we need to drastically change the definitions that are used under the one-port model without overlap [24, 3]: see the example in Section 3.3.1 below.

3.3 Mapping Strategies

Key metrics for a given workflow are the throughput and the latency. The throughput measures the aggregate rate of processing of data, and it is the rate at which data sets can enter the system. Equivalently, the inverse of the throughput, defined as the period, is the time interval required between the beginning of the execution of two consecutive data sets. The latency is the time elapsed between the beginning and the end of the execution of a given data set, hence it measures the response time of the system to process the data set entirely. Note that minimizing the latency is antagonistic to minimizing the period

The mapping problem consists of assigning application stages to platform processors. Formally, we search for an allocation function of stages to processors, defined as $a : [1..n] \to [1..p]$. We always assume in the following that $a(0) = in$ and $a(n+1) = out$.

There are several mapping strategies. The more restrictive mappings are ***one-to-one***; in this case, each stage is assigned a different processor. Then the allocation function a is a one-to-one function, and there must be at least as many processors as application stages. Another strategy is very common for linear chains: we may decide to group consecutive stages onto a same processor, in order to avoid some costly communications. However, a processor is only processing an interval of consecutive stages. Such a mapping is called an ***interval-based*** mapping. Finally, we can consider ***general mappings***, for which there is no constraint on the allocation function: each processor is assigned one or several stage intervals.

3.3.1 Working out an Example

Consider the little example of Figure 2 with four stages. Below each stage S_i we have indicated the number of computations w_i (expressed in flops) that it requires: $w_1 = 2$, $w_2 = 1$, $w_3 = 3$ and $w_4 = 4$. The value of each δ_i is indicated at the right of each stage: $\delta_0 = 1$, $\delta_1 = \delta_2 = 4$, $\delta_3 = \delta_4 = 1$. As for the platform, assume that we have a *Fully Homogeneous* platform with two identical processors P_1 and P_2 of speed $s = 1$ and of network card capacities $\mathsf{B}^i = \mathsf{B}^o = 1$, and with identical links of bandwidth $b = 1$.

We can achieve a perfect load-balance of the computations if we map stages S_1 and S_3 on P_1, and stages S_2 and S_4 on P_2. What would be the period and the latency with such a mapping? Under the

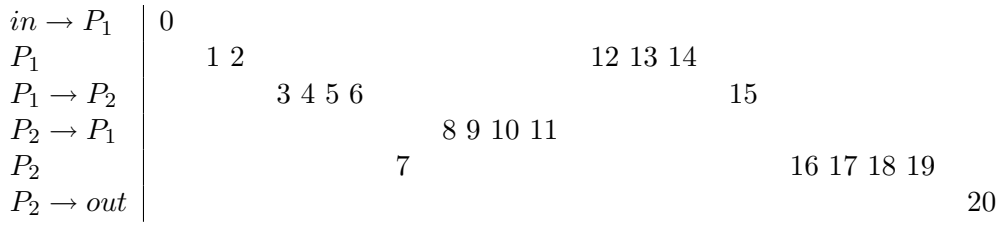


Figure 3: Processing the first data set to achieve a latency $\mathcal{L} = 21$ would lead to conflicts for the next data sets.

bounded multiport model with computation/communication overlap, we achieve a period $\mathcal{P} = 5$. Indeed, P_1 has two incoming communications of size $\frac{\delta_0}{b} + \frac{\delta_2}{b} = 5$, and we have $\frac{\delta_0+\delta_2}{\mathsf{B}^i} = 5$, so that all constraints (link bandwidths and network card capacity) are verified for incoming communications. Similarly, we check that for outgoing communications $\frac{\delta_1}{b} + \frac{\delta_3}{b} = \frac{\delta_1+\delta_3}{\mathsf{B}^o} = 5$. Finally, we chose the mapping so that $\frac{w_1+w_3}{s} = 5$. Altogether, the cycle-time of processor P_1 is 5, which means that it can start to process a new data set every 5 time units. We perform the same analysis for P_2 and derive that its cycle-time also is 5. The period is the maximum of the cycle-times of the processors, hence we derive that $\mathcal{P} = 5$.

Computing the latency is more complicated. At first sight, we might say that the latency is the longest path in the execution, of length

$$\frac{\delta_0}{b} + \frac{w_1}{s} + \frac{\delta_1}{b} + \frac{w_2}{s} + \frac{\delta_2}{b} + \frac{w_3}{s} + \frac{\delta_3}{b} + \frac{w_4}{s} = 21$$

However, this is only possible for the first data sets. See Figure 3: if the first data set $\mathsf{ds}^{(0)}$ enters the platform at time $t = 0$, then P_1 is active at time $t = 1$ and $t = 2$ (for stage S_1), and then $t = 12$, $t = 13$, and $t = 14$ (for stage S_3). If a new data set enters every five time-units to achieve a period $\mathcal{P} = 5$, then data set $\mathsf{ds}^{(k)}$ enters at time $t = 5k$ and P_1 is active for it at time $t = 1 + 5k, 2 + 5k, 12 + 5k, 13 + 5k, 15 + 5k$. Because this holds true for all $k \geq 0$, we have many conflicts! For instance the first conflict is at $t = 12$ for stage S_1 of $\mathsf{ds}^{(3)}$ and stage S_3 of $\mathsf{ds}^{(1)}$. Similarly, we obtain conflicts for P_2 and for the communication link from P_1 to P_2.

In fact, in steady-state we can only achieve a much larger latency: see Figure 4, where the latency is shown to be $\mathcal{L} = 45$. The idea is simple: when a processor executes some computation for a data set $\mathsf{ds}^{(k)}$, then it simultaneously receives input corresponding to data set $\mathsf{ds}^{(k+1)}$ and performs output corresponding to data set $\mathsf{ds}^{(k-1)}$. In turn, the next processor operates on data set $\mathsf{ds}^{(k-2)}$, and so on. This example shows that a key parameter is the number of stage intervals in the mapping. Here we define an interval as a subset of consecutive stages that is not mapped onto the same processor as the previous stages (see Section 3.3.3 for a more formal definition). In our example we have four intervals composed of 1 stage each, because each stage is mapped onto a different processor than its predecessor, hence $K = 4$. If there are K intervals, there are $K + 1$ communication links, hence it takes $K + (K + 1) = 2K + 1$ periods for a data set to be processed entirely. We check in Figure 4 that we need $2K + 1 = 9$ periods to compute a data set, so that $\mathcal{L} = 9 \times 5 = 45$.

Note that computing the latency with a non-overlap model looks very difficult. This is because each processor would have to decide which of its two incoming communications to execute first, and which of its two outgoing communications to execute first. Any choice is likely to increase the

	...	period k	period $k+1$	period $k+2$	...
$in \to P_1$	...	$\mathsf{ds}^{(k)}$	$\mathsf{ds}^{(k+1)}$	$\mathsf{ds}^{(k+2)}$	...
P_1	...	$\mathsf{ds}^{(k-1)}$, $\mathsf{ds}^{(k-5)}$	$\mathsf{ds}^{(k)}$, $\mathsf{ds}^{(k-4)}$	$\mathsf{ds}^{(k+1)}$, $\mathsf{ds}^{(k-3)}$	...
$P_1 \to P_2$	...	$\mathsf{ds}^{(k-2)}$, $\mathsf{ds}^{(k-6)}$	$\mathsf{ds}^{(k-1)}$, $\mathsf{ds}^{(k-5)}$	$\mathsf{ds}^{(k)}$, $\mathsf{ds}^{(k-4)}$	...
$P_2 \to P_1$	...	$\mathsf{ds}^{(k-4)}$	$\mathsf{ds}^{(k-3)}$	$\mathsf{ds}^{(k-2)}$	...
P_2	...	$\mathsf{ds}^{(k-3)}$, $\mathsf{ds}^{(k-7)}$	$\mathsf{ds}^{(k-2)}$, $\mathsf{ds}^{(k-6)}$	$\mathsf{ds}^{(k-1)}$, $\mathsf{ds}^{(k-5)}$	...
$P_2 \to out$	...	$\mathsf{ds}^{(k-8)}$	$\mathsf{ds}^{(k-7)}$	$\mathsf{ds}^{(k-6)}$	...

Figure 4: Achieving a latency $\mathcal{L} = 9\mathcal{P} = 45$ in steady state mode.

latency for some data set. The problem is similar for the one-port or the multiport model. Indeed, with the one-port model, we have no choice and must serialize the two communications. On the contrary, with the multiport model we can decide to execute both communications in parallel, but this is not helpful as it only delays the first communication without speeding up the second one. For both models it is hard to decide which communication to give priority to. A simple (greedy) algorithm would give priority to the communication involving the least recent data set. We could easily work out such an algorithm for our little example. However, computing a closed form expression for the latency in the general case seems untractable. Furthermore, we point out that without overlap the difficulty comes from the fact that several stage intervals are mapped onto the same processor, which requires to arbitrate between as many incoming (and outgoing) communications. If we enforce *interval-based* mappings (see Section 3.3), then each processor is assigned a single interval and the computation for the latency is greatly simplified.

Note also that *interval-based* mappings are interesting for the multiport model as they allow to decrease the value of K, the number of stage intervals, which never exceeds the number p of available processors for such mappings. However, general mappings may still be needed to better balance the work, hence to decrease the period, at the price of a larger value of K. Such trade-offs are at the heart of the algorithms and complexity proofs that follow. Finally, we point out that introducing feedback loops in Section 8 will further complicate these issues.

3.3.2 Period

As illustrated in Figure 4, we assume that a new data set arrives every period at a regular pace. Let P_1 be the processor in charge of the first stage. While it is computing for data set k, it simultaneously receives input corresponding to data set $k+1$ and sends output corresponding to data set $k-1$. More precisely, the latter output is related to the first stage S_i such that $a(i) \neq a(i+1)$: for a given data set, all computations corresponding to stages S_1 to S_i are performed during the same period. As in the example of Figure 2, P_1 can be assigned other stage intervals, and the period must be large enough so that the sum of all its computations does not exceed the value of the period. The same holds true for the sum of its incoming communications, and for the sum of its outgoing communications.

Formally, under the bounded multiport model with overlap, the cycle-time of processor P_u,

$1 \leq u \leq p$, is defined as:

$$\mathcal{P}(u) = \max \begin{cases} \displaystyle\sum_{1 \leq k \leq n \ \& \ a(k)=u} \frac{w_k}{s_u} & (1) \\ \max \left(\displaystyle\max_{1 \leq v \leq p, v \neq u} \displaystyle\sum_{\substack{1 \leq k \leq n \\ a(k)=u \ \& \ a(k-1)=v}} \frac{\delta_{k-1}}{b_{v,u}}, \displaystyle\sum_{\substack{1 \leq k \leq n \\ a(k)=u \ \& \ a(k-1) \neq u}} \frac{\delta_{k-1}}{\mathsf{B}_u^i} \right) & (2) \\ \max \left(\displaystyle\max_{1 \leq v \leq p, v \neq u} \displaystyle\sum_{\substack{1 \leq k \leq n \\ a(k)=u \ \& \ a(k+1)=v}} \frac{\delta_k}{b_{u,v}}, \displaystyle\sum_{\substack{1 \leq k \leq n \\ a(k)=u \ \& \ a(k+1) \neq u}} \frac{\delta_k}{\mathsf{B}_u^o} \right) & (3) \end{cases}$$

Line (1) is bounding the period when the computation step is the longest activity. It expresses the computation time for processor P_u. Line (2) represents the time for input communications, which is bounded by the links from incoming processors, and by the network card limit B_u^i. We need to consider all stages S_k that are assigned to P_u but whose predecessor S_{k-1} is not. We check that no link bandwidth is exceeded from any other processor P_v and we account for all these communications together for the network card capacity of P_u. Similarly, line (3) deals with output communications. Also, notice that for the sake of simplicity, $in = 0$ and $out = n + 1$.

The period of the mapping is then $\mathcal{P} = \max_{1 \leq u \leq p} \mathcal{P}(u)$.

3.3.3 Latency

As stated in Section 3.3.1, the latency depends on the number of stage intervals, or equivalently, of the number of changes from one processor to a different one.

We define $K(i, j)$ as the number of stage intervals between stages S_i and S_j, where $i < j$. Formally,

$$K(i, j) = \sum_{\substack{i \leq k < j \\ a(k) \neq a(k+1)}} 1$$

The total number of intervals in the pipeline is $K = K(1, n + 1)$. Since we always have $a(n + 1) = out \neq a(n)$, $K \geq 1$ ($K = 1$ if all stages are mapped onto the same processor $P_{a(n)}$). Again, we point out that K depends on the number of processor changes, and thus it is increased if a processor is in charge of several distinct stage intervals.

The latency is then defined as $2K + 1$ times the period, since a data set traverses the whole pipeline in $2K + 1$ time-steps, and each time-step has a duration of $\mathcal{P}$:

$$\mathcal{L} = (2K + 1) \times \mathcal{P}$$

Consider again the toy example of Figure 2. With the mapping that was chosen, we had $\mathcal{P} = 5$ but $K = 9$, hence $\mathcal{L} = 45$. The value of the period is optimal because the sum of the four computation weights is 10, and we have two processors of speed 1. But the value of the latency is not optimal. For instance if we assign all stages to the same processor, the period becomes $\mathcal{P} = 10$ but now $K = 3$, hence $\mathcal{L} = 30$. We can also assign the first three stages to P_1 and the last one to P_2: then we derive $\mathcal{P} = 6$, $K = 5$ and $\mathcal{L} = 30$ too.

4 Complexity Results

This section provides complexity results for period and latency minimization. We first prove that minimizing period is NP-complete for *Fully Homogeneous* platforms. Then we provide an example where a non-interval mapping minimizes the latency on *Fully Homogeneous* platform. This leads us to conjecture that latency minimization is also NP-complete for *Fully Homogeneous* platforms. Unfortunately, we have not been able to prove this result. However, we do prove that latency minimization is NP-complete on *Communication Homogeneous* platforms.

Theorem 1. *On Fully Homogeneous platforms, finding the general mapping which minimizes the period is NP-complete.*

Proof. We consider the associated decision problem: given a bound B on the period, is there a mapping of period less than or equal to B? The problem is obviously in NP: given a period and a mapping, it is easy to check in polynomial time whether it is valid or not.

To establish the completeness, we use a reduction from 2-PARTITION [10]. We consider an instance $\mathcal{I}_1$ of 2-PARTITION: given m positive integers $a_1, a_2, \ldots, a_m$, does there exist a subset $I \subset \{1, \ldots, m\}$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$. Let $S = \sum_{i=1}^{m} a_i$.

We build the following instance $\mathcal{I}_2$ of our problem: the workflow is a linear chain composed of m stages S_i with $w_i = a_i$ $(1 \leq i \leq m)$, and $p = 2$ processors of speed 1. There are no communications. We ask whether it is possible to realize a period less than or equal to $B = S/2$. Clearly, the size of $\mathcal{I}_2$ is polynomial (and even linear) in the size of $\mathcal{I}_1$. We now show that instance $\mathcal{I}_1$ has a solution if and only if instance $\mathcal{I}_2$ does.

Suppose first that $\mathcal{I}_1$ has a solution, I. Since there are no communications, the period of a mapping is only determined by the size of stages allocated to each processor. We assign stages $S_i, i \in I$ to processor 1, and the remaining stages to the other processor. The period of both processors is thus $S/2$.

On the other hand, if $\mathcal{I}_2$ has a solution, then we define I as the set of indices of stages allocated to processor 1. By definition of the period, and since the speed is equal to 1, $\sum_{i \in I} a_i \leq B = S/2$, and the period on processor 2 imposes that $\sum_{i \notin I} a_i \leq S/2$. Since $\sum_{i=1}^{m} a_i = S$, both previous sums equal $S/2$, and we found a solution to $\mathcal{I}_1$. □

Like the period minimization problem, we believe that latency minimization is also NP-complete. In Section 5, we shall see that optimal interval mappings can be found in polynomial time. However, unfortunately, interval mappings are not guaranteed to be optimal for latency either. The following example shows that interval mappings are not optimal for latency for *Fully Homogeneous* platforms. Consider 150 homogeneous processors with speeds all equal to 1. The application pipeline contains 300 stages as shown below, and there are no communications.

$$99 \quad 1 \quad \underbrace{100 \quad 1}_{\times 148} \quad 101 \quad 1$$

In other words, the first stage has work 99, the second stage has work 1. The third and the fourth stages have work 100 and 1 respectively. These third and fourth stages are repeated 148 times (the fifth stage has work 100, sixth stage 1, and so on). The 299-th stage has work 101 and the last stage has work 1 again.

The best schedule is perfectly load balanced but not interval-based: put stages 1,2, and 300 on one processor, all pairs of 100 and 1 on distinct 148 processors, and then stage 299 with work 101 on the last processor. The period of this mapping is 101 and the latency is 30603. The best interval-based schedule uses 75 intervals with period 203, and latency 30653 (we used the dynamic programming algorithm of Section 5 to obtain this result).

Since minimum latency mapping may not be an interval-based mapping, it appears as though latency minimization is in fact NP-complete for *Fully Homogeneous* platforms. However, the proof has not been forthcoming, and this problem is left open. The next theorem shows that minimizing latency is NP-complete for *Communication Homogeneous* platforms.

Theorem 2. *On Communication Homogeneous platforms, finding the general mapping which minimizes the latency is NP-complete.*

Proof. Recall that the latency for a general mapping is defined as

$$\mathcal{L} = (2q+1)\mathcal{P}_q,$$

where q is the number of intervals in the mapping, and $\mathcal{P}_q$ the period, i.e., the maximum cycle-time of any processor involved in the execution. If a given processor is assigned several (non-consecutive) intervals, its cycle-time is the maximum of the three following costs: (i) sum of its incoming communications (one per interval); (ii) sum of its outgoing communications (one per interval); and (iii) sum of its computations (all its assigned stages).

Let Latency-Dec denote the decision problem: given an application pipeline with n stages, a *Communication Homogeneous* platform with p processors, and a bound L, does there exist a mapping whose latency is not greater than L? The Latency-Dec problem clearly belongs to the class NP: given a mapping of the stages onto p, it is easy to compute which intervals are mapped on the same processor, and to compute the maximum cycle-time $\mathcal{P}_q$ of all the p processors and to check that $(2q+1)\mathcal{P}_q \leq \mathsf{L}$.

To establish the completeness, we use a reduction from NUMERICAL MATCHING WITH TARGET SUMS (NMWTS), which is NP-complete in the strong sense [10]. We consider an instance $\mathfrak{I}_1$ of NMWTS: given $3m$ numbers $x_1, x_2, \ldots, x_m$, $y_1, y_2, \ldots, y_m$ and $z_1, z_2, \ldots, z_m$, does there exist two permutations σ_1 and σ_2 of $\{1, 2, \ldots, m\}$, such that $x_i + y_{\sigma_1(i)} = z_{\sigma_2(i)}$ for $1 \leq i \leq m$? Because NMWTS is NP-complete in the strong sense, we can encode the $3m$ numbers in unary and assume that the size of $\mathfrak{I}_1$ is $O(m+M)$, where $M = \max_i\{x_i, y_i, z_i\}$. We also assume that $\sum_{i=1}^m x_i + \sum_{i=1}^m y_i = \sum_{i=1}^m z_i$, otherwise $\mathfrak{I}_1$ cannot have a solution.

We build the following instance $\mathfrak{I}_2$ of Latency-Dec:

- We define $n = (M+3)m$ stages, whose computation weights w_i are outlined below:

$$A_1 \quad \underbrace{111\ldots1}_{M} \quad C \quad D \quad | \quad A_2 \quad \underbrace{111\ldots1}_{M} \quad C \quad D \quad | \quad \ldots \quad | \quad A_m \quad \underbrace{111\ldots1}_{M} \quad C \quad D$$

  here, we introduce $U = 72m^2$ (see below where this value comes from) and we let $B = (U+2)M$, $C = B + 3M$, $D = B + 5M$, and $A_i = B + x_i$ for $1 \leq i \leq m$. To define the w_i formally for $1 \leq i \leq n$, let $N = M + 3$. We have for $1 \leq i \leq m$:

$$\begin{cases} w_{(i-1)N+1} = A_i = B + x_i \\ w_{(i-1)N+j} = 1 \text{ for } 2 \leq j \leq M+1 \\ w_{iN-1} = C, \qquad w_{iN} = D \end{cases}$$

- For the communication weights, we let $\delta_i = 1$ for all the n stages
- We define a platform with $p = 3m$ processors and homogeneous links of bandwidth $b = 1$, As for the speeds, we let s_i be the speed of processor P_i where, for $1 \leq i \leq m$:

$$s_i = B + z_i, \qquad s_{m+i} = C + M - y_i, \qquad s_{2m+i} = D$$

Finally, we ask whether there exists a solution matching the bound $\mathsf{L} = 6m + 1$. Clearly, the size of $\mathfrak{I}_2$ is polynomial in the size of $\mathfrak{I}_1$. We now show that instance $\mathfrak{I}_1$ has a solution if and only if instance $\mathfrak{I}_2$ does.

Suppose first that $\mathfrak{I}_1$ has a solution, with permutations σ_1 and σ_2 such that $x_i + y_{\sigma_1(i)} = z_{\sigma_2(i)}$. For $1 \leq i \leq m$:

- We map each stage A_i and the following $y_{\sigma_1(i)}$ stages of weight 1 onto processor $P_{\sigma_2(i)}$.
- We map the following $M - y_{\sigma_1(i)}$ stages of weight 1 and the next stage, of weight C, onto processor $P_{m+\sigma_1(i)}$.
- We map the next stage, of weight D, onto processor P_{2m+i}.

We have a partition of all the stages into $p = 3m$ intervals. For $1 \leq i \leq m$, the load and speed of the processors are indeed equal:

- The load of $P_{\sigma_2(i)}$ is $A_i + y_{\sigma_1(i)} = B + x_i + y_{\sigma_1(i)}$ and its speed is $B + z_{\sigma_2(i)}$.
- The load of $P_{m+\sigma_1(i)}$ is $M - y_{\sigma_1(i)} + C$, which is equal to its speed.
- The load and speed of P_{2m+i} are both D.

Each processor is assigned a single interval, hence the cost of its incoming and outgoing communications is equal to 1. The mapping achieves a period $P_p = 1$, and a latency $\mathcal{L} = (2p + 1)P_p = 6m + 1 = \mathsf{L}$, hence a solution to $\mathfrak{I}_2$.

Suppose now that $\mathfrak{I}_2$ has a solution, i.e., a mapping matching the latency bound $\mathsf{L} = 6m + 1$. We first observe that all the p processors must be enrolled in the execution. Indeed, if only $q < p$ processors participate, the period P_q verifies

$$P_q \geq \frac{\mathsf{W_{total}}}{q\mathsf{s_{max}}}$$

Here, $\mathsf{W_{total}} = \sum_{i=1}^{n} w_i = m(3U + 15)M + \sum_{i=1}^{m} x_i$ is the sum of all stage weights, and $\mathsf{s_{max}} = D$ is the largest processor speed. For the latency, we derive that

$$\mathcal{L} = (2q + 1)P_q \geq m\frac{2q + 1}{q}\frac{3U + 15}{U + 7} \geq 3m\left(2 + \frac{1}{3m - 1}\right)\frac{U + 5}{U + 7}$$

We rewrite this expression as

$$\mathcal{L} \geq 6m + 1 + \frac{1}{3m - 1} - \left(6m + 1 + \frac{1}{3m - 1}\right)\frac{2}{U + 7}$$

and finally

$$\mathcal{L} \geq 6m + 1 + \frac{1}{6m - 2} > \mathsf{L}$$

for $U = 72m^2$ (hence this mysterious value of U). Now we have p processors and therefore at least p intervals. But if we had more than p intervals, one processor would be responsible for at least two intervals, and its cycle-time would be at least 2 because of incoming communications. As a consequence, the latency would be at least

$$\mathcal{L} \geq (2p + 3).2 > \mathsf{L}$$

Hence we have p intervals in the mapping, one per processor. We derive that the period P_p does not exceed 1 (because $\mathsf{L} = (2p+1)P_p$).

Next we observe that $s_i < s_{m+j} < s_{2m+k} = D$ for $1 \leq i, j, k \leq m$. Indeed $s_i = B + z_i \leq B + M = (U + 3M)$, $(U+5)M \leq s_{m+j} = C + M - y_j \leq (U+6)M$ and $D = (U+7)M$. Hence each of the m stages of weight D must be assigned to a processor of speed D, and it is the only stage assigned to this processor. These m singleton assignments divide the set of stages into m intervals, namely the set of stages before the first stage of weight D, and the $m-1$ sets of stages lying between two consecutive stages of weight D. The total weight of each of these m intervals is $A_i + M + C > B + M + C = (2U+8)M$, while the largest speed of the $2m$ remaining processors is at most $U + 6M$. Therefore each of them must be assigned to at least 2 processors each. However, there remains only $2m$ available processors, hence each interval is assigned exactly 2 processors.

Consider such an interval A_i 111...1 C with M stages of weight 1, and let P_{i_1} and P_{i_2} be the two processors assigned to this interval. Stages A_i and C are not assigned to the same processor (otherwise the whole interval would). So P_{i_1} receives stage A_i and h_i stages of weight 1 while P_{i_2} receives $M - h_i$ stages of weight 1 and stage C. The weight of P_{i_2} is $M - h_i + C \geq C = (U+5)M$ while $s_i \leq (U+3)M$ for $1 \leq i \leq m$. Hence P_{i_1} must be some P_i, $1 \leq i \leq m$ while P_{i_2} must be some P_{m+j}, $1 \leq j \leq m$. Because this holds true on each interval, this defines two permutations $\sigma_2(i)$ and $\sigma_1(i)$ such that $P_{i_1} = P_{\sigma_2(i)}$ and $P_{i_2} = P_{\sigma_1(i)}$. Because $P_p \leq 1$, we have:

- $A_i + h_i = B + x_i + h_i \leq B + z_{\sigma_2(i)}$
- $M - h_i + C \leq C + M - y_{\sigma_1(i)}$

Therefore $y_{\sigma_1(i)} \leq h_i$ and $x_i + h_i \leq z_{\sigma_2(i)}$. But by hypothesis, $\sum_{i=1}^m x_i + \sum_{i=1}^m y_i = \sum_{i=1}^m z_i$, hence all these inequalities are tight: we have $y_{\sigma_1(i)} = h_i$ and $x_i + y_{\sigma_1(i)} = z_{\sigma_2(i)}$ for all i. Altogether, we have found a solution for $\mathfrak{I}_1$, which concludes the proof. □

Corollary 1. *On Communication Homogeneous platforms, finding an interval mapping which minimizes the latency is NP-complete.*

Proof. See the proof for Theorem 2. □

5 Algorithms for Interval-Based Mappings

This section gives a polynomial time algorithm returning the best interval-based mapping, both for period and latency minimization problems on a *Fully Homogeneous* platform.

Theorem 3. *On a Fully Homogeneous platform, finding the interval-based mapping which minimizes the period or the latency can be constructed in polynomial time.*

Proof. We provide a dynamic programming to build an interval-based mapping which minimizes the period or the latency. Let $P(i,u)$ be the best period that can be achieved by mapping stages S_1 to S_i onto at most u processors. The recurrence writes as follows:

$$P(i,u) = \min_{0 \leq j < i} \left\{ \max \left\{ P(j, u-1), \frac{\sum_{k=j+1}^{i} w_k}{s}, \frac{\delta_i}{b} \right\} \right\}$$

with the initializations, for $u \geq 0$ and $i > 0$:

$$P(0,u) = \frac{\delta_0}{b} \quad \text{and} \quad P(i,0) = +\infty.$$

In fact, we either map all the stages onto the same processor ($j = 0$ in the min), or we cut the stages at an arbitrary place ($0 < j < i$) and allocate stages $j + 1$ to i to a single processor. The period is then the maximum between computations ($\sum w_k$) and communications (output δ_i/b). The input communication is taken into account in the term $P(j, u - 1)$, since the output from stage j is identical to the input to stage $j+1$. The first input communication is counted as $P(0, u)$. If there are not enough processors and we still have stages to process, we reach the case $P(i, 0)$ with $i > 0$ which is not feasible, thus leading to a period of $+\infty$.

For the period minimization problem, we compute $P(n, p)$, as any solution uses at most p processors.

For the latency minimization problem, we need to compute $\min_{1 \leq K \leq p}(2K + 1).P(n, K)$. The latency depends on the number of cuts, so it may be better to use less processors to decrease the $2K + 1$ value. Notice that if the solution returned by $P(n, K)$ uses strictly less than K processors, the result will be reproduced with a smaller K, leading to a better latency, and thus the first solution will not be taken into account. □

6 Approximation Algorithms for Period and Latency

This section describes approximation algorithms to minimize the period and latency of a linear chain application. Theorem 4 proves that some interval-based mapping provides a 2-approximation of the best general mapping for the period of a pipeline. Theorem 5 proves that mapping all the stages on the fastest processor provides a 1.5-approximation for optimal latency for *Communication Homogeneous* platforms.

Theorem 4. *Some interval-based mapping provides a 2-approximation for the optimal period for Fully Homogeneous platforms.*

Proof. In this proof, given an optimal mapping, we generate an interval-based mapping that has at most twice the period.

Consider the best general mapping with k intervals. If the best mapping has fewer than p intervals, then we are done, since there is an equivalent interval-based mapping which returns the optimal period. In fact, we cannot increase the period by giving each processor a distinct interval, and we have enough processors to do so.

Hence let us consider the case where $k > p$. Let the period of this optimal mapping be P_o. For every interval i, where $1 \leq i \leq k$, let Δ_i be the incoming communication, and Σ_i be the computation requirements of interval i (Δ_{i+1} is the outgoing communication for stage i). Without loss of generality, assume that for all processors u, we have $s_u = \mathsf{B}^i_u = \mathsf{B}^o_u = 1$, and for all processor pairs u, v, $b_{u,v} = 1$. Therefore, for all i, we have $\Delta_i \leq P_o$, and $\Sigma_i \leq P_o$. In addition, we know that $\sum_{i=1}^{k} \Sigma_i \leq pP_o$, since that is the maximum amount of computation p processors can perform in P_o time.

Now, we generate an interval-based mapping. We merge the consecutive intervals of the general mapping into larger intervals, called *partitions*. If a partition r contains intervals j to interval l, then $\sum_{i=j}^{l-1} \Sigma_i < P_o$ and $\sum_{i=j}^{l} \Sigma_i \geq P_o$. The first partition starts at interval 1, and the last one ends at interval k. There are at most p partitions, since the total computation is $\sum_{i=1}^{k} \Sigma_i \leq pP_o$ and each partition does at least P_o computation.

Since we know that all intervals i have $\Sigma_i \leq P_o$, we know that for any partition r that contains intervals j to l, we have $\sum_{i=j}^{l} \Sigma_i \leq 2P_o$. In addition, the incoming communication for partition r

is $\Delta_r = \Delta_j \leq P_o$ and the outgoing communication for partition r is $\Delta_{r+1} = \Delta_{l+1} \leq P_o$. Therefore, for every partition r, we have $\Sigma_r \leq 2P_o$, $\Delta_r \leq P_o$ and $\Delta_{r+1} \leq P_o$. Therefore, each of these partitions can be mapped on different processors, to get an interval-based mapping with period at most $2P_o$. □

Corollary 2. *Some interval-based mapping provides a 2-approximation for the optimal latency for Fully Homogeneous platforms.*

Proof. This corollary is a direct consequence of Theorem 4, but we shall not prove it since we provide a better approximation algorithm for latency in Theorem 5. □

Unfortunately, interval mappings are not a constant-approximation for the period on *Communication Homogeneous* platforms. Here is an example to demonstrate this fact. Consider a *Communication Homogeneous* platform with $n+1$ processors; processor 1 has speed $s_1 = 2K$ and the other n processors have speed 1, and $n > K^2$. Now consider an application as follows:

$$K \quad \underbrace{1}_{\times n} \quad K$$

In other words, the first and the last stages have a computation of K, and there are n intermediate stages which have a computation of 1. In this example, a non-interval schedule would map the first and the last stages to processor 1, and the remaining stages (one each) to the n processors with speed 1, generating a period of 1. Unfortunately, any interval schedule has a period of at least K since $n > K^2$. Therefore, the interval schedule can be as bad as a K-approximation at best for the period.

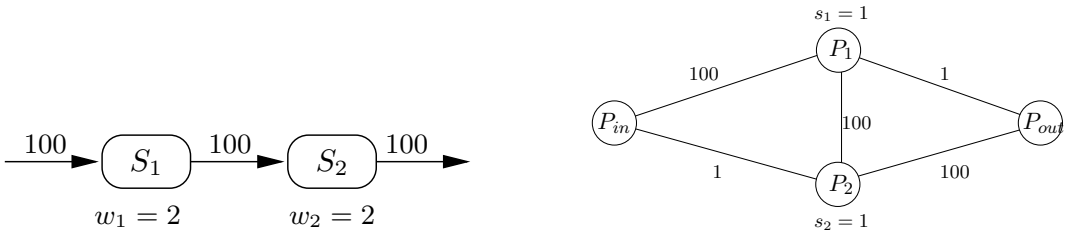
Theorem 5. *A mapping which puts all the stages on the fastest processor provides a 1.5-approximation for optimal latency on Communication Homogeneous platforms.*

Proof. Consider the optimal mapping, which uses k processors. Say that the latency with this mapping is L_o, and the period with this mapping is P_o. Since there are at least k intervals in this mapping, we have $L_o \geq (2k+1)P_o$. Now let the latency and period when all the stages are mapped on one processor be L_1 and P_1 respectively. We know that $P_1 \leq kP_o$. Since $L_1 = 3P_1$, we have $L_1 = 3P_1 \leq 3kP_o \leq 3kL_o/(2k+1) \leq 1.5L_o$. □

Unfortunately, mapping all stages on the fastest processor does not provide a constant approximation for optimum latency for *Fully Heterogeneous* platforms, see Figure 5 for counterexample. If we map both stages on either processor, then the period of this processor is 100 due to the slow communication link connecting it to either P_{in} or P_{out}. Therefore, the latency is 300. However, if we map S_1 on P_1 and S_2 on P_2, then the period of both processors is 2, and the latency is 10.

7 Special Cases: Infinite Number of Processors and Uniform Applications

Since both period and latency minimization appear to be NP-complete even on *Fully Homogeneous* platforms, we consider some special cases where one can in fact find polynomial time algorithms. The two special cases we consider are are when there is an infinite number of processors, and when the application is uniform.



(a) Application. (b) Platform.

Figure 5: Example where mapping all stages on the fastest processor does not provide a 1.5-approximation for latency.

7.1 Infinite Number of Processors

We start with a preliminary lemma which shows that there is an optimal interval-based mapping when considering an infinity of homogeneous processors.

Lemma 1. *With an infinite number of processors on a Fully Homogeneous platform, there is an optimal interval-based mapping for both problems, (i) minimizing the period, and (ii) minimizing the latency.*

Proof. This can be proved easily with an exchange argument. If processor P_u is not handling a single interval of stages, since we have an infinite number of processors, we can give each interval of stages processed by P_u to a new processor. Both the period and the latency can only decrease, since the new processors have less computation to perform than P_u, communications have not changed, and the number of distinct intervals in the mapping K is still the same. □

The following theorem proves that both optimal period and latency can be found in polynomial time with an infinite number of processors.

Theorem 6. *With an infinite number of processors on a Fully Homogeneous platform, finding the general mapping which minimizes the period or the latency can be done in polynomial time.*

Proof. This theorem is a direct consequence from Lemma 1 and Theorem 3. Indeed, Theorem 3 provides the best interval-based mapping, and Lemma 1 ensures that this interval-based mapping is better than any general mapping. □

Also we notice that with a fixed number of processors, still on *Fully Homogeneous* platforms, Lemma 1 is not true anymore. Indeed, we can build an example in which the best latency is obtained with a general mapping which is not interval-based. Please refer to Section 4 for such an example.

7.2 Uniform Applications

Now, let us consider a regular application, i.e., $\forall 1 \leq i \leq n,\ w_i = w$ and $\delta_i = \delta$, on a *Communication Homogeneous* platform. In this particular case, we have a lemma similar to Lemma 1: there is always an optimal mapping for latency which is interval-based.

Lemma 2. *For a regular application on a Communication Homogeneous platform, there is an optimal interval-based mapping for both problems, (i) minimizing the period, and (ii) minimizing the latency.*

Proof. This can be easily proved using an exchange argument: if the optimal mapping is not interval-based, we can keep the same computational workload to each processor, but grouped on a single interval. The period can only decrease (we reduce potentially the amount of communications), and K can only decrease, since we reduce the number of cuts. Thus, the latency can also only decrease. □

Building on this Lemma, we can write a dynamic programming algorithm to find the optimal period and latency in polynomial time.

Theorem 7. *For a regular application on a Communication Homogeneous platform, finding the interval-based mapping which minimizes the period or the latency can be done in polynomial time.*

Proof. We provide a dynamic programming to build an interval-based mapping which minimizes the period or the latency. Let $P(i, u)$ be the best period that can be achieved by mapping stages S_1 to S_i onto at most u processors. Processors are renumbered by increasing speeds, thus $s_u \geq s_{u-1} \geq ... \geq s_1$. Of course, if $u < p$, we keep only the u fastest processors. The recurrence writes as follows:

$$P(i,u) = \min_{0 \leq j < i} \left\{ \max \left\{ P(j, u-1), (i-j).\frac{w}{s_u}, \frac{\delta}{b} \right\} \right\}$$

with the initializations, for $u \geq 0$ and $i > 0$:

$$P(0,u) = \frac{\delta}{b} \quad \text{and} \quad P(i,0) = +\infty.$$

The recurrence is similar to that of Theorem 3, except the computation time is now $(i-j)$ times a single computation on the fastest processor, since all stages are identical.

For the period minimization problem, we compute $P(n, p)$, as any solution uses at most p processors. For the latency minimization problem, we need to compute $\min_{1 \leq K \leq p}(2K+1).P(n, K)$, as in Theorem 3. □

8 Additional Complexity of Feedback Loops

Most of previous problems are already NP-hard, except some particular cases. Some of these special polynomial cases become NP-difficult when adding the extra complexity of feedback loops, as for instance the period minimization problem with an infinite number of processors. Also, some of the approximation results do not hold anymore.

Before revisiting previous complexity results, we formalize the feedback loops model and explain how period and latency should be computed so as to take feedback loops into account.

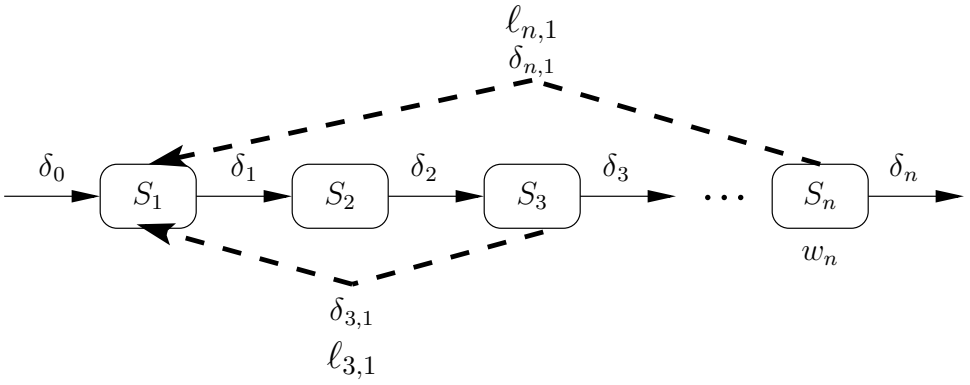


Figure 6: Feedback loops.

8.1 Model for Feedback Loops

There might be some dependencies between data sets for different stages, represented as feedback loops, see Figure 6 for an example with two feedback loops. An arrow going from stage $S_{k'}$ to stage S_k, where $k' > k$, and labeled with a positive integer $\ell_{k',k}$, means that S_k needs the output from $S_{k'}$ of data set $i - \ell_{k',k}$ to compute data set i. The size of data to be transferred along this feedback arrow is denoted as $\delta_{k',k}$. Such an arrow generates a loop in the application graph.

For each feedback loop, the feedback data must arrive on time so that it is possible to perform the desired computation on the next data. A mapping will thus be valid only if there are not too many stage intervals (or processor changes) inside a loop.

Assume that there is a loop labeled with $\ell_{j,i}$, going from S_j to S_i (with $j > i$). As discussed in Section 3.3, processor $a(i)$ is processing data set $\mathsf{ds}^{(k)}$ while sending data set $\mathsf{ds}^{(k-1)}$ to the next processor in the line, which in the meantime processes data set $\mathsf{ds}^{(k-2)}$ and sends data set $\mathsf{ds}^{(k-3)}$, and so on. Thus, in order to get the data set on time, we need to ensure that $2.(K(i,j) - 1 + \Delta_{a(i)\neq a(j)}) \leq \ell_{j,i}$, where $\Delta_{a(i)\neq a(j)} = 1$ if $a(i) \neq a(j)$, and 0 otherwise.

For instance, consider the feedback loop from S_3 to S_1 in Figure 6. If stages S_1 and S_2 are mapped onto, say, processor P_1 while S_3 is mapped onto P_2, then $K = 2$, $a(1) = 1 \neq a(3) = 2$, and the formula states that we must have $4 \leq \ell_{3,1}$. Indeed, when P_1 operates on data set $\mathsf{ds}^{(k)}$, P_2 operates on $\mathsf{ds}^{(k-2)}$ and sends data corresponding to $\mathsf{ds}^{(k-3)}$ back to P_1, just in time if $\ell_{3,1} = 4$ for P_1 to compute $\mathsf{ds}^{(k+1)}$ during the next period.

Note that if the whole interval from S_i to S_j is mapped onto the same processor, then $K = 1$ and $\Delta_{a(i)\neq a(j)} = 0$, hence we derive the constraint $0 \leq \ell_{j,i}$, which is fine because data is available on site for the next period.

Feedback loops not only impose constraints on the mapping. We also need to revisit the expression for the period that was given in Section 3.3.2 to account for the additional communications induced by the feedback loops. Rather than going on formally, we just illustrate this with the previous example: with the same mapping, the feedback loop from S_3 to S_1 induces an additional input communication that must be added to the other incoming communications of P_1, and an additional output communication that must be added to the other outgoing communications of P_1.

The generalization of interval-based mappings when considering feedback loops are *connected-subgraph* mappings, in which each processor is assigned a connected subgraph instead of an interval.

Thus, two stages linked with a feedback loop can be mapped on the same processor, so that the feedback communication is done locally.

8.2 Infinite Number of Processors

Theorem 8. *On Fully Homogeneous platforms with an infinite number of processors, finding the general mapping which minimizes the period for a linear chain application graph with feedback loops is NP-complete.*

Proof. We consider the associated decision problem: given a period P, is there a mapping of period less than P? The problem is obviously in NP: given a period and a mapping, it is easy to check in polynomial time whether it is valid or not.

To establish the completeness, we use a reduction from 2-PARTITION [10]. We consider an instance $\mathcal{I}_1$ of 2-PARTITION: given n positive integers $a_1, a_2, \ldots, a_n$, does there exist a subset $I \subset \{1, \ldots, n\}$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$. Let $S = \sum_{i=1}^{n} a_i$.

We build the following instance $\mathcal{I}_2$ of our problem: the workflow is a linear chain composed of $n+1$ stages S_i with $w_1 = 1$ and $w_i = a_{i-1}$ $(2 \leq i \leq n+1)$. All communications between pipeline stages are identical: $\delta_i = 1$ for $0 \leq i \leq n+1$. Then we add one feedback loop per stage $S_i, i > 1$, going back to S_1. The communication costs for feedback loops are defined as $\delta_{i,1} = a_{i-1}$ for $2 \leq i \leq n+1$, and the labels on the loop are not constraining the period: $\ell_{i,1} = 3n$. Processor speeds, bandwidths and network card capacities are identical: $s = b = \mathsf{B}^i = \mathsf{B}^o = 1$. Figure 7 illustrates this application.

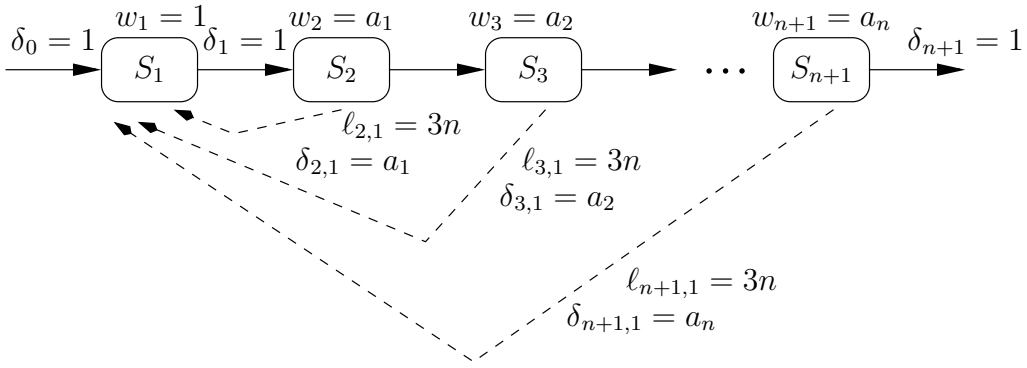


Figure 7: Application used in the reduction for the period with an infinite number of processors with feedback loops.

Then we ask whether it is possible to realize a period of $S/2 + 1$. Clearly, the size of $\mathcal{I}_2$ is polynomial (and even linear) in the size of $\mathcal{I}_1$. We now show that instance $\mathcal{I}_1$ has a solution if and only if instance $\mathcal{I}_2$ does.

Suppose first that $\mathcal{I}_1$ has a solution, I. We assign stages $S_{i+1}, i \in I$ and S_1 to the same processor, P_1 and all remaining stages to distinct processors $P_{i+1}, i \notin I$. The only source of incoming communications for processor P_{i+1} is the previous stage, thus a total of $1 \leq S/2 + 1$. Its computation time is $a_i \leq S/2 + 1$, and output communication $1 + a_i \leq S/2 + 1$, including the output to next stage and output through the feedback loop. For processor P_1, the total output

communication is only 1, but its input is $1 + \sum_{i \notin I} a_i = S/2 + 1$ since it must receive the feedback data from all stages not mapped onto the same processor. Its computation is $1 + \sum_{i \in I} a_i = S/2 + 1$, adding computations of all stages mapped onto P_1. Thus the achieved period is $S/2 + 1$.

On the other hand, if $\mathcal{I}_2$ has a solution, let P_1 be the processor on which S_1 is mapped. P_1 may handle other stages; I is the set of indices of stages allocated to P_1. If we consider the computation done by P_1, we must have $1 + \sum_{i \in I} a_{i-1} \leq 1 + S/2$, and thus $\sum_{i \in I} a_{i-1} \leq S/2$. The incoming communications to P_1 should not exceed the period either, thus $1 + \sum_{i \notin I} a_{i-1} \leq 1 + S/2$, and $\sum_{i \notin I} a_{i-1} \leq S/2$. Since $\sum_{2 \leq i \leq n+1} a_{i-1} = S$, both equations can be satisfied only if $\sum_{i \in I} a_{i-1} = \sum_{i \notin I} a_{i-1} = S/2$. Therefore we found a solution to $\mathcal{I}_1$. □

Like period, interval mappings are not optimal for latency on infinite number of processors when the application has feedback loops. It seems unlikely that the mapping with minimum latency can be found in polynomial time, since this problem appears to be as difficult as the problem of minimizing latency while mapping on a finite number of processors. We have, however, not been able to prove that this problem is indeed NP-complete.

8.3 Approximation Results

Also, the approximation result for period on *Fully Homogeneous* platforms (Theorem 4) does not hold when adding feedback loops. Here we give a counterexample that shows that interval mapping cannot provide any constant-approximation for pipeline workflows with feedback loops.

Consider a long pipeline with n stages, where each stage i has $w_i = 1$. The last stage has a feedback loop back to the first stage, as shown in Figure 8. All the communication requirements between stages (except on the feedback link) are 1. On the other hand, the feedback link has a communication requirement of X. Given a platform with $n-1$ identical processors (with processor speeds and bandwidths of 1), a non-interval mapping will map the first and the last stage on the same processor, giving a period of 2, and all the other stages on distinct processors. Any interval mapping that maps the first and the last stage on different processors has a period of X. Similarly, if $n > X$, then the period is greater than X if all the stages are mapped on the same processor. Therefore, interval mappings are not constant-competitive for *Fully Homogeneous* platforms when the application has feedback loops.

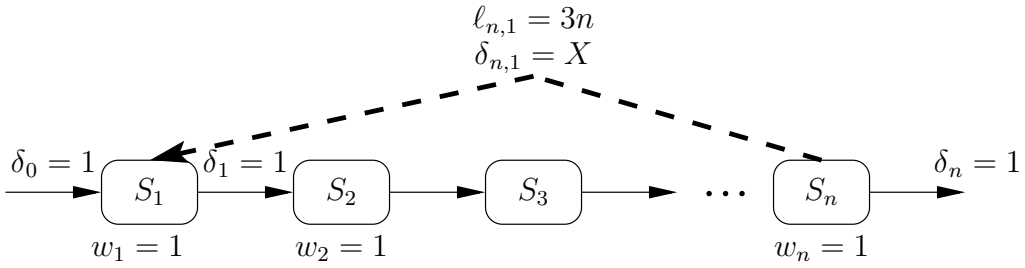


Figure 8: Counter example that shows that interval mappings are not constant-competitive for period when the application contains feedback loops.

The approximation result for latency on *Communication Homogeneous* platforms still holds for applications with feedback loops using the same proof given in Theorem 5 in Section 6.

9 Conclusion

This work presents complexity results for mapping linear workflows with computation/communication overlap, under the bounded multiport model. We provide a formal definition of period and latency optimization problems for such linear workflows and prove several major results; in particular the NP-completeness of the period minimization problem even on *Fully Homogeneous* platforms. Latency becomes NP-complete as soon as platforms are *Communication Homogeneous*, and the complexity remains open for *Fully Homogeneous* platforms. We provide a 2-approximation algorithm for the period on *Fully Homogeneous* platforms and a 1.5-approximation algorithm for the latency on *Communication Homogeneous* platforms. For some special mapping rules (restricting to interval-based mappings) and special cases (infinite number of identical processors, regular applications), we succeed in deriving polynomial algorithms for *Fully Homogeneous* platforms. Also, we introduce the concept of feedback loops to provide control to linear workflows. Such feedbacks add a level of complexity to most previous problems, since some special cases become NP-complete, and approximation results for the period do not hold anymore.

We believe that this exhaustive study of complexity results provides a solid theoretical foundation for the study of linear workflow mappings, with or without feedback loops, under the bounded multiport model with overlap. As future work, we plan to design some efficient polynomial-time heuristics to solve the many combinatorial instances of the problem, and to assess their performance through extensive simulations. This can become challenging in the presence of feedback loops. Finally, it would be interesting to study workflows in a different context (like web-service applications [22]) where each stage S_i has a ***selectivity*** σ_i parameter which is the ratio between its input and output data δ_{i-1}/δ_i. In these problems, the application dag is not fixed, and the aim is to generate the dag and schedule it so as to decrease the period and/or latency. There may be some constraints on what types of dags are permissible. For instance, given two stages S_1 and S_2 with selectivities $\sigma_1 < \sigma_2$, on a homogeneous platform, it is better to create a dag where S_1 precedes S_2 to minimize the period. We plan to generalize our results for these kinds of applications.

References

[1] M. Banikazemi, V. Moorthy, and D. K. Panda. Efficient collective communication on heterogeneous networks of workstations. In *Proceedings of the 27th International Conference on Parallel Processing (ICPP'98)*. IEEE Computer Society Press, 1998.

[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 heterogeneous platforms. In *HeteroPar'2004: International Conference on Heterogeneous Computing, jointly published with ISPDC'2004: International Symposium on Parallel and Distributed Computing*, pages 296–302. IEEE Computer Society Press, 2004.

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

[4] M. Beynon, A. Sussman, U. Catalyurek, T. Kurc, and J. Saltz. Performance optimization for data intensive grid applications. In *PProceedings of the Third Annual International Workshop on Active Middleware Services (AMS'01)*. IEEE Computer Society Press, 2001.

[5] M. D. Beynon, T. Kurc, A. Sussman, and J. Saltz. Optimizing execution of component-based applications using group instances. *Future Generation Computer Systems*, 18(4):435–448, 2002.

[6] P. Bhat, C. Raghavendra, and V. Prasanna. Efficient collective communication in distributed heterogeneous systems. In *ICDCS'99 19th International Conference on Distributed Computing Systems*, pages 15–24. IEEE Computer Society Press, 1999.

[7] P. Bhat, C. Raghavendra, and V. Prasanna. Efficient collective communication in distributed heterogeneous systems. *Journal of Parallel and Distributed Computing*, 63:251–263, 2003.

[8] 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.

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

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

[11] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, Sept. 1996. see also `http://www-unix.mcs.anl.gov/mpi/mpich/`.

[12] 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.

[13] 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.

[14] N. T. Karonis, B. Toonen, and I. Foster. Mpich-g2: A grid-enabled implementation of the message passing interface. *J.Parallel and Distributed Computing*, 63(5):551–563, 2003.

[15] S. Khuller and Y. Kim. On broadcasting in heterogenous networks. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1011–1020. Society for Industrial and Applied Mathematics, 2004.

[16] B. Kreaseck, L. Carter, H. Casanova, J. Ferrante, and S. Nandy. Interference-aware scheduling. *International Journal of High Performance Computing Applications*, 20(1):45–59, 2006.

[17] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4):406–471, 1999.

[18] P. Liu. Broadcast scheduling optimization for heterogeneous cluster systems. *Journal of Algorithms*, 42(1):135–152, 2002.

[19] 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.

[20] T. Saif and M. Parashar. Understanding the behavior and performance of non-blocking communications in MPI. In *Proceedings of Euro-Par 2004: Parallel Processing*, LNCS 3149, pages 173–182. Springer, 2004.

[21] M. Spencer, R. Ferreira, M. Beynon, T. Kurc, U. Catalyurek, A. Sussman, and J. Saltz. Executing multiple pipelined data analysis operations in the grid. In *2002 ACM/IEEE Supercomputing Conference*. ACM Press, 2002.

[22] U. Srivastava, K. Munagala, J. Widom, and R. Motwani. Query optimization over web services. In *VLDB '06: Proceedings of the 32nd Int. Conference on Very Large Data Bases*, pages 355–366. VLDB Endowment, 2006.

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

[24] 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.

[25] 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.

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

[27] 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.

[28] H. Topcuoglu, S. Hariri, and M. Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distributed Systems*, 13(3):260–274, 2002.

[29] 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`.

[30] 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/2007`.

[31] T. Yang and A. Gerasoulis. DSC: Scheduling parallel tasks on an unbounded number of processors. *IEEE Trans. Parallel and Distributed Systems*, 5(9):951–967, 1994.