2008 14th IEEE International Conference on Parallel and Distributed Systems

Mapping Linear Workflows with Computation/Communication Overlap

Kunal Agrawal
CSAIL, Massachusetts Institute of Technology, USA
kunal_ag@mit.edu

Anne Benoit and Yves Robert
LIP, École Normale Supérieure de Lyon, France
{Anne.Benoit|Yves.Robert}@ens-lyon.fr

Abstract

This paper presents theoretical results for mapping and scheduling linear workflows onto heterogeneous platforms. We use a realistic architectural model, representative of current multi-threaded systems. Our model has bounded communication capabilities and full computation/communication overlap. In these workflow applications, the goal is often to maximize throughput or to minimize latency. We present several complexity results, and approximation algorithms, for these two criteria. We also consider the implications of adding feedback loops to linear chain applications.

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 are being continually designed to support these applications. In these languages, the programmer expresses 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 NP-complete; therefore, we consider the special case of linear chains with possibly feedback edges. 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. We provide comprehensive theoretical analysis of the complexity of mapping these graphs on both homogeneous and heterogeneous distributed memory platforms.

Subhlok and Vondran [10] 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 [2]. In this model, at any time, the processor can either compute, receive an incoming communication, or send an outgoing communication. 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. 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 explore its complexity before extending the results 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

1521-9097/08 $25.00 © 2008 IEEE
DOI 10.1109/ICPADS.2008.107

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 for all platforms in the bounded multiport model with overlap. 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 results are stated in Section 3, together with an optimal polynomial algorithm to find the best ***interval-based mapping*** for both throughput and latency on *Fully Homogeneous* platforms. These interval mappings are commonly used since they minimize the communication overhead. However, finding the interval-based mapping with optimal throughput or latency is NP-complete for *Communication Homogeneous* platforms.

• Since finding the best mapping is NP-complete, we present approximation algorithms; namely in Section 4, we show that interval-based 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. 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-based mappings are optimal for throughput, hence we have polynomial time algorithms. See Section 5 for this result.

• In Section 6, we show that almost all problems become more difficult when feedback loops are added. Interval-based 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.

Due to lack of space, proofs have not been included. In addition, some of the related work has not been mentioned in this section. Please refer to the extended version [1] for full proofs and more related work.

2. Framework

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

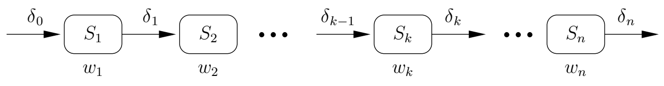

Figure 1. Application linear chain.

2.1. Application model

We consider simple application workflows whose graphs are a pipeline (i.e., a linear chain). 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 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.

2.2. Execution model

2.2.1 Platform graph

We target a heterogeneous platform with p processors $P_u, 1 \leq u \leq p$, fully interconnected as a (virtual) clique. 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. There is a bidirectional link $link_{u,v} : P_u \rightarrow P_v$ between any processor pair P_u and P_v, of bandwidth $b_{u,v}$. We use 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. Note that we do not need to have a physical link between all pairs of processors. We may have a switch, or a path composed of several physical links, instead, to interconnect P_u and P_v; in the latter case the $b_{u,v}$ is the bandwidth of the slowest link in the path. In addition to link bandwidths, we have processor network cards that 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 B_u^i data items per time-unit, and it cannot send more than B_u^o data items per time-unit. In the most general case, we have fully heterogeneous platforms, with different processors speeds, link capacities and network card capacities.

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., $P_{in} = P_{out}$.

We classify particular cases which are important, both from a theoretical and practical perspective. *Fully Homogeneous* platforms have identical processors ($s_u = s$) and homogeneous communication devices ($b_{u,v} = b$ for link bandwidths, and $\mathsf{B}_u^i = \mathsf{B}^i$, $\mathsf{B}_u^o = \mathsf{B}^o$ for network cards). They represent typical parallel machines. *Communication Homogeneous* platforms are still interconnected with homogeneous communication devices, but they have different-speed processors ($s_u \neq s_v$). They correspond to networks of workstations with plain TCP/IP interconnects or other LANs. *Fully Heterogeneous* platforms are the most general, fully heterogeneous architectures. Hierarchical platforms made up with several clusters interconnected by slower backbone links can be modeled this way.

2.2.2 Realistic communication models

The standard model for DAG scheduling heuristics [7] 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 [3], where a given processor can be involved in a single communication at any time-step, either a send or a receive. 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 [8]. The one-port model fully accounts for the heterogeneity of the platform, as each link has a different bandwidth. A study of mapping strategies for linear chain application graphs under the one-port model has been conducted in [2].

Another realistic model is the *bounded multiport* model [5]. 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 [6] now allow for initiating multiple concurrent send and receive operations, thereby providing practical realizations of the multiport model.

2.2.3 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 operating system are indeed capable of such an overlap.

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 [10, 2]: see the example in Section 2.3.1 below.

2.3. 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. 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 $a : [1..n] \rightarrow [1..p]$. 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.

2.3.1 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 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 $\frac{\delta_0}{b} + \frac{w_1}{s} + \frac{\delta_1}{b} + \frac{w_2}{s} + \frac{\delta_2}{b} + \frac{w_3}{s} + \frac{\delta_3}{b} + \frac{w_4}{s} + \frac{\delta_4}{b} = 21$. 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 $t = 1+5k, 2+5k, 12+5k, 13+5k, 14+5k$. 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

$$\begin{array}{ccccccccc} \xrightarrow{1} & S_1 & \xrightarrow{4} & S_2 & \xrightarrow{4} & S_3 & \xrightarrow{1} & S_4 & \xrightarrow{1} \\ & 2 & & 1 & & 3 & & 4 & \end{array}$$

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

$in \rightarrow P_1$	0
P_1	1 2 12 13 14
$P_1 \rightarrow P_2$	3 4 5 6 15
$P_2 \rightarrow P_1$	8 9 10 11
P_2	7 16 17 18 19
$P_2 \rightarrow out$	20

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

obtain conflicts for P_2 and for the link from P_1 to P_2.

In fact, in steady-state we can only achieve a much larger latency if period $\mathcal{P}$ should not be exceeded: see Figure 4, where the latency is $\mathcal{L} = 45$. 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. An interval is a subset of consecutive stages that is not mapped onto the same processor as the previous stages (see Section 2.3.3 for a more formal definition). Here 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 under period constraints with a non-overlap model looks very difficult. Each processor would have to decide which of its two

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

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

incoming communications to execute first, and which of its two outgoing communications to execute first. Any choice is likely to increase the 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.

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 then each processor is assigned a single interval and the computation for the latency is greatly simplified. Interval mappings are also 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 results that follow. Finally, we point out that introducing feedback loops in Section 6 will further complicate these issues.

2.3.2 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 $\mathcal{P}(u)$ of processor P_u, $1 \leq u \leq p$, is defined as the maximum of $\sum_{1 \leq k \leq n \ \& \ a(k)=u} \frac{w_k}{s_u}$ (1),
$\max_{1 \leq v \leq p, v \neq u} \sum_{\substack{1 \leq k \leq n \\ a(k)=u \\ a(k-1)=v}} \frac{\delta_{k-1}}{b_{v,u}}$ (2), $\sum_{\substack{1 \leq k \leq n \\ a(k)=u \\ a(k-1) \neq u}} \frac{\delta_{k-1}}{\mathsf{B}^i_u}$ (3),
$\max_{1 \leq v \leq p, v \neq u} \sum_{\substack{1 \leq k \leq n \\ a(k)=u \\ a(k+1)=v}} \frac{\delta_k}{b_{u,v}}$ (4), and $\sum_{\substack{1 \leq k \leq n \\ a(k)=u \\ a(k+1) \neq u}} \frac{\delta_k}{\mathsf{B}^o_u}$ (5).

The first term is bounding the period when the computation step is the longest activity. It expresses the computation time for processor P_u. The second and third terms represent the time for input communications, which is bounded by the links from incoming processors (2), and by the network card limit B^i_u (3). 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, (4) and (5) deal with output communications. Also, notice that for the sake of simplicity, $in = 0$ and $out = n+1$.

The period of the mapping is then

$$\mathcal{P} = \max_{1 \leq u \leq p} \mathcal{P}(u) \qquad (1)$$

2.3.3 Latency

As outlined in Section 2.3.1, the latency cannot be defined independently of the period since we enforce that the constraint related to the period of the mapping should be fulfilled. More precisely, we proceed as follows: given a mapping, we first compute its period $\mathcal{P}$ according to equation (1). Next we count 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, $K(i,j) = \sum_{i \leq k < j \ \& \ a(k) \neq a(k+1)} 1$. 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 finally 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 $\mathcal{P}$:

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

Note that [4] uses the formula $\mathcal{L} = (2K-1) \times \mathcal{P}$ because input/output is not taken into account.

Consider again the example of Figure 2. With the chosen mapping, $\mathcal{P} = 5$ but $K = 9$, hence $\mathcal{L} = 45$. The value of the period is optimal (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: if we assign all stages to the same processor, the period becomes $\mathcal{P} = 10$ but $K = 3$, hence $\mathcal{L} = 30$. We can also assign the first three stages to P_1 and the last one to P_2: we derive $\mathcal{P} = 6$, $K = 5$ and $\mathcal{L} = 30$ too.

3. Complexity results

This section provides complexity results for period and latency minimization.

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

Like the period minimization problem, we believe that latency minimization is also NP-complete. In Section 3, we shall see that optimal interval-based 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 mapping 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 mapping uses 75 intervals with period 203, and latency 30653 (we used the dynamic programming algorithm of Theorem 3 to obtain this result [1]).

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.*

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

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 using a dynamic programming algorithm.*

4. Approximation algorithms

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

Unfortunately, interval-based 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-based mapping 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-based mapping has a period of at least K since $n > K^2$. Therefore, the interval-based mapping 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.*

Unfortunately, mapping all stages on the fastest processor does not provide a constant approximation for optimum latency for *Fully Heterogeneous* platforms, see [1] for a counter-example.

5. Infinite number of 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.*

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.*

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 3 for such an example.

6. 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.

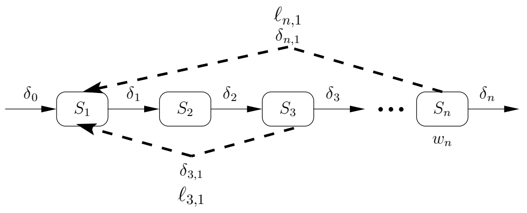

Figure 5. Feedback loops.

6.1. Model for feedback loops

There might be some dependencies between data sets for different stages, represented as feedback loops, see Figure 5 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 2.3, 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 5. 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 2.3.2 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_2. 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.

6.2. Infinite Number of Processors

Theorem 7. *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.*

Similarly to period, interval-based 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.

6.3. Approximation Results

Also, the approximation result for period on *Fully Homogeneous* platforms (Theorem 4) does not hold when adding feedback loops. Indeed, we show in [1] that interval-based mappings cannot provide any constant-approximation for pipeline workflows with feedback loops. However, 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 4.

7. 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), 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. It would also be interesting to study workflows in a different context (like web-service applications [9]) 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. 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] K. Agrawal, A. Benoit, and Y. Robert. Mapping linear workflows with computation/communication overlap. Research Report 2008-21, LIP, ENS Lyon, France, June 2008. Available at `graal.ens-lyon.fr/~abenoit/`.

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

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

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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.