On the Complexity of Mapping Linear Chain Applications onto Heterogeneous Platforms

Anne Benoit, Yves Robert, Eric Thierry

On the Complexity of Mapping Linear Chain Applications onto Heterogeneous Platforms

Anne Benoit, Yves Robert and Eric Thierry

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

October 2008

LIP Research Report RR-2008-32

Abstract

In this paper, we explore the problem of mapping simple application patterns onto large-scale heterogeneous platforms. An important optimization criteria that should be considered in such a framework is the latency, or makespan, which measures the response time of the system in order to process one single data set entirely. We focus in this work on linear chain applications, which are representative of a broad class of real-life applications. For such applications, we can consider one-to-one mappings, in which each stage is mapped onto a single processor. However, in order to reduce the communication cost, it seems natural to group stages into intervals. The interval mapping problem can be solved in a straightforward way if the platform has homogeneous communications: the whole chain is grouped into a single interval, which in turn is mapped onto the fastest processor. But the problem becomes harder when considering a fully heterogeneous platform. Indeed, we prove the NP-completeness of this problem. Furthermore, we prove that neither the interval mapping problem nor the similar one-to-one mapping problem can be approximated by any constant factor (unless P=NP).

Key words: pipeline graphs, interval mappings, latency, makespan, complexity results, NP-hardness, approximation.

1 Introduction

Mapping applications onto parallel platforms is a difficult challenge. Several scheduling and load-balancing techniques have been developed for homogeneous architectures (see [8] for a survey) but the advent of heterogeneous clusters has rendered the mapping problem even more difficult.

In this context, a structured programming approach rules out many of the problems which the low-level parallel application developer is usually confronted to, such as deadlocks or process starvation. Moreover, many real applications draw from a range of well-known solution paradigms, such as pipelined or farmed computations. High-level approaches based on algorithmic skeletons [4, 6] identify such patterns and seek to make it easy for an application developer to tailor such a paradigm to a specific problem. A library of skeletons is provided to the programmer, who can rely on these already coded patterns to express the communication scheme within its own application. Moreover, the use of a particular skeleton carries with it considerable information about implied scheduling dependencies, which we believe can help address the complex problem of mapping a distributed application onto a heterogeneous platform.

In this paper, we consider applications that can be expressed as pipeline graphs. Typical applications include digital image processing, where images have to be processed in steady-state mode. A well known pipeline application of this type is for example JPEG encoding (http://www.jpeg.org/). In such workflow applications, a series of data sets (tasks) enter the input stage and progress from stage to stage until the final result is computed. Each stage has its own communication and computation requirements: it reads an input file from the previous stage, processes the data and outputs a result to the next stage. For each data set, initial data is input to the first stage, and final results are output from the last stage. One of the key metrics for such applications is the latency, i.e., 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.

Due to the possible local memory accesses, the rule of the game is always to map a given stage onto a single processor: we cannot process half of the tasks on a processor and the remaining tasks on another without exchanging intra-stage information, which might be costly and difficult to implement. In other words, a processor that is assigned a set of stages will execute the operations required by these stages (input, computation and output) for all the tasks fed into the pipeline. Communications are paid only if two consecutive stages are not mapped onto the same processor, and the latency is the sum of computation and communication costs over the whole pipeline.

The optimization problem can be stated informally as follows: which stage to assign to which processor in order to minimize the latency? 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 which assigns a processor to each stage 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 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.

The problem of mapping pipeline skeletons onto parallel platforms has received some attention. In particular, Subhlok and Vondran [9, 10] have dealt with this problem on homogeneous platforms. In this paper, we extend their work and target heterogeneous platforms. Our main goal is to assess the additional complexity induced by the heterogeneity of processors and of communication links.

The rest of the paper is organized as follows. Section 2 is devoted to the presentation of the target optimization problems. Next in Section 3 we proceed to the complexity results, and in particular we prove the NP-completeness of the interval mapping problem. Furthermore, we prove that both this problem and the one-to-one mapping problem cannot be approximated by any constant factor (unless P=NP). Finally, we state some concluding remarks in Section 4.

2 Framework

2.1 Application Model

The application is expressed as a pipeline graph of n stages $\mathcal{S}_k$, $1 \leq k \leq n$, as 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 $\mathcal{S}_k$ 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. The first stage $\mathcal{S}_1$ receives an input of size δ_0 from the outside world, while the last stage $\mathcal{S}_n$ returns the result, of size δ_n, to the outside world.

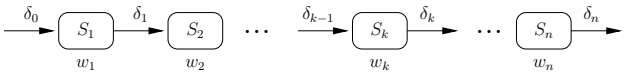


Figure 1: The application pipeline.

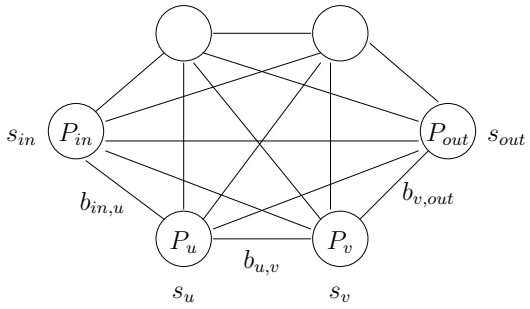


Figure 2: The target platform.

2.2 Platform Model

We target a platform (see Figure 2), with m 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 $b_{u,v}$. 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 (or receive) a message of size X from P_u to P_v.

Communication contention is taken care of by enforcing the *one-port* model [2, 3]. 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 [7].

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}.

2.3 Mapping Problem

The general mapping problem consists in assigning application stages to platform processors. For simplicity, we could assume that each stage $\mathcal{S}_i$ of the application pipeline is mapped onto a distinct processor (which is possible only if $n \leq p$). However, such one-to-one mappings may be unduly restrictive, and a natural extension is to search for interval mappings, i.e., allocation functions where each participating processor is assigned an interval of consecutive stages. Intuitively, assigning several consecutive tasks to the same processor will increase its computational load, but may well dramatically decrease communication requirements. In fact, the best interval mapping may turn out to be a one-to-one mapping, or instead may enroll only a very small number of fast computing processors interconnected by high-speed links. Interval mappings constitute a natural and useful generalization of one-to-one mappings (not to speak of situations where $m < n$, where interval mappings are mandatory), and such mappings have been studied by Subhlock et al. [9, 10].

Formally, we search for a partition of $[1..n]$ into $m \leq p$ intervals $I_j = [d_j, e_j]$ such that $d_j \leq e_j$ for $1 \leq j \leq m$, $d_1 = 1$, $d_{j+1} = e_j + 1$ for $1 \leq j \leq m-1$ and $e_m = n$. The allocation function $a(j)$ returns the index of the processor on which interval I_j is mapped. The optimization problem is to determine the mapping with the minimum latency, over all possible partitions into intervals, and over all processor assignments.

We assume that $a(0) = in$ and $a(m+1) = out$, where P_{in} is a special processor holding the initial data, and P_{out} is receiving the results. The latency is then obtained by summing all communication and computation costs for a given allocation of intervals to processors:

$$\mathcal{L} = \frac{\delta_0}{b_{in,a(1)}} + \sum_{1 \leq j \leq m} \left\{ \frac{\sum_{i=d_j}^{e_j} w_i}{s_{a(j)}} + \frac{\delta_{e_j}}{b_{a(j),a(j+1)}} \right\} \tag{1}$$

With no constraint on the allocation function, the mapping is *general* (or arbitrary): a processor is allowed to process several non consecutive intervals. In order to restrict to *interval* mappings, we request the allocation function to be one-to-one, so that all intervals are mapped on distinct processors:

$$\forall 1 \leq j_1, j_2 \leq m \quad a(j_1) = a(j_2) \Longrightarrow j_1 = j_2 \tag{2}$$

For *one-to-one* mappings, we add the additional constraint that each interval is reduced to exactly one stage ($d_j = e_j$ for $1 \leq j \leq m$).

3 Complexity Results

In this section, we summarize complexity results for the latency minimization problem, with the different mapping rules. We start by recalling results for platforms with homogeneous communication links (but different-speed processors), before tackling fully heterogeneous platforms.

3.1 Homogeneous Communications

When the target platform has identical communication links, i.e., $b_{u,v} = b$ for $1 \leq u, v \leq p$, then latency can be minimized by grouping all application stages into a single interval, and mapping this interval onto the fastest processor, P_u. This is clearly an optimal mapping:
- according to Equation (1), all computation costs are minimized thanks to the use of P_u;
- the only communications are δ_0/b and δ_n/b, and these are included in any mapping.

For one-to-one mappings, all communication costs always are included, with a total cost $\frac{\sum_{i=0}^{n} \delta_i}{b}$. Thus we need to minimize the total computation cost. This can be done greedily by assigning the stage with the largest computation cost to the fastest processor, and so on. A simple exchange argument proves the optimality of this mapping. Thus we have the following theorem:

Theorem 1. *Minimizing the latency is polynomial on communication homogeneous platforms for one-to-one, interval and general mappings.*

However, this line of reasoning does not hold anymore when communications become heterogeneous: indeed, the communication cost can hugely differ from one mapping to another, as shown in the next Section.

3.2 Heterogeneous Platforms

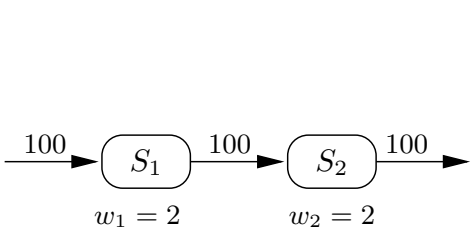
3.2.1 Motivating Example

Consider the problem of mapping the pipeline of Figure 3(a) on the heterogeneous platform of Figure 3(b). The pipeline consists of two stages, both needing the same amount of computation ($w = 2$), and the same amount of communications ($\delta = 100$). In this example, a mapping which minimizes the latency must map each stage on a different processor, thus splitting the stages into two intervals. In fact, if we map the whole pipeline on a single processor, we achieve a latency of $100/100 + (2 + 2)/1 + 100/1 = 105$, either if we choose P_1 or P_2 as target processor. Splitting the pipeline and hence mapping the first stage on P_1 and the second stage on P_2 requires to pay the communication between P_1 and P_2 but drastically decreases the latency: $100/100 + 2/1 + 100/100 + 2/1 + 100/100 = 1 + 2 + 1 + 2 + 1 = 7$. This little example explains why minimizing the latency cannot be longer achieved by mapping all stages onto the fastest resource.

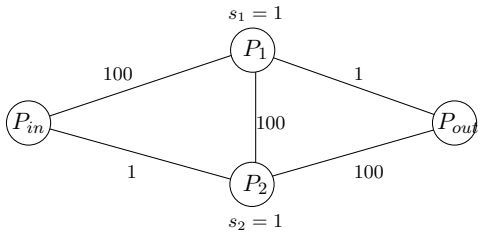
3.2.2 About One-to-One and General Mappings

Theorem 2. *Minimizing the latency is NP-hard on heterogeneous platforms for one-to-one mappings.*

This problem can be reduced from the Traveling Salesman Problem (TSP), which is NP-complete [5]. The proof can be found in [1].



(a) Example optimal with 2 intervals.



(b) The pipeline has to be split into intervals to achieve an optimal latency on this platform.

Figure 3: Motivating Example.

Theorem 3. *Minimizing the latency is polynomial on heterogeneous platforms for general mappings.*

General mappings are less constrained since several stages can be arbitrarily grouped onto a same processor. In this case, the problem can be reduced to finding a shortest path in a graph, or solved by using dynamic programming. The proof using shortest paths can be found in [1], and we provide here a dynamic program to solve this problem.

Let $L(i, u)$ be the minimum latency that can be achieved by mapping stage S_i onto processor P_u. Then the recurrence can be written as:

$$L(i,u) = \min_{v \neq u} \begin{cases} L(i-1,u) + \frac{w_i}{s_u} & (1) \\ L(i-1,v) + \frac{\delta_{i-1}}{b_{v,u}} + \frac{w_i}{s_u} & (2) \end{cases}$$

Line (1) corresponds to the case in which stage $i-1$ is mapped on the same processor as stage i, thus only the computation cost of S_i is added to the latency. Line (2) tries all other processors $P_v \neq P_u$ and adds to the latency the corresponding communication cost (plus the computation cost for S_i).

The initialization must ensure that the correct communication cost will be paid as an input to S_1. This can be done by forcing the virtual stage S_0 to be mapped onto P_{in}:

$$L(0,u) = \begin{cases} 0 & \text{if } P_u = P_{in} \\ +\infty & \text{otherwise} \end{cases}$$

Finally, we need to compute

$$\min_{1 \leq u \leq p} L(n,u) + \frac{\delta_n}{b_{u,out}}$$

The complexity of this algorithm is $O(n.p^2)$. The result mapping is *general* because nothing prevents to map non-consecutive stages onto a same processor. If we look for an *interval* mapping, we can modify the algorithm in order to keep track of the processors already used, for instance by marking processor P_u as "used" when switching to processor P_v in line (2) of the recurrence. However, such an information cannot be handled without having an algorithm of exponential complexity. Indeed, we prove below that the interval mapping problem is NP-hard.

3.2.3 NP-Completeness for Interval Mappings

Definition 1. LATENCY-INT-HET-DEC– *Given a pipeline application, a target platform and a bound L, does there exist an interval mapping of the pipeline on the platform with a latency which does not exceed L?*

Theorem 4. LATENCY-INT-HET-DEC *is NP-complete.*

Proof. The problem clearly belongs to NP: given a mapping, we can check that constraint (2) is fulfilled (hence we do have an interval mapping), compute its latency using Equation (1), and then check that the bound L is respected, all of this in polynomial time.

To prove the completeness of the problem, we use a reduction from DISJOINT-CONNECTING-PATH (DCP), which is NP-complete [5]. Consider an arbitrary instance $\mathcal{I}_1$ of DCP, i.e., a graph $G = (V, E)$ and a collection of $k+1$ disjoint vertex pairs $(x_1, y_1), (x_2, y_2), ..., (x_{k+1}, y_{k+1}) \in V^2$. We ask whether G contains $k+1$ mutually vertex-disjoint paths, one connecting x_i and y_i for each i, $1 \leq i \leq k+1$. The number of nodes in the graph is $n = |V|$, and we have $k \leq n$.

We build the following instance $\mathcal{I}_2$ of the latency minimization problem.

• The application consists in $n(k+1)$ stages, whose computation costs are outlined below:

$$\underbrace{w^k ... w^k}_{n-2} \quad w^{2k} \quad \epsilon^k \quad \underbrace{w^{k-1} ... w^{k-1}}_{n-2} \quad w^{2k-1} \quad \epsilon^{k-1} \quad \dots \quad \underbrace{w ... w}_{n-2} \quad w^{k+1} \quad \epsilon \quad \underbrace{1 ... 1}_{n}$$

Formally, for $0 \leq i \leq k-1$, the $n-2$ stages $in+1$ to $in+n-2$ have a computation cost w^{k-i}, stage $in+n-1$ has a computation cost w^{2k-i}, and stage $in+n$ has a computation cost ϵ^{k-i}. Finally, the n last stages have a computation cost 1. The value of ϵ is small, it is fixed to ensure that some slow processors can only process the stages of cost ϵ^i, for $1 \leq i \leq k$. Thus, $\epsilon = \frac{1}{L+1}$, where L is the target latency which will be defined later. On the other hand, w is large: we fix $w = n^3$. All communication costs δ_i are identical: $\delta_i = \delta = \frac{1}{n+k+1}$.

• The target platform is composed of $n+k$ processors, one corresponding to each vertex of the initial graph of DCP, and k additional processors $z_i, 1 \leq i \leq k$.

Processor speeds are all equal to 1 except for processors z_i, $1 \leq i \leq k$, which are "fast" processors of speed w^{k-i+1}, and processors x_i, $2 \leq i \leq k+1$, which are "slow" processors of speed ϵ^{k-i+2}.

The bandwidth between two processors $(n_i, n_j) \in V^2$ equals 1 if $(n_i, n_j) \in E$, and $\frac{\delta}{L}$ otherwise (where L is the target latency). Thus, using a communication link which does not correspond to an edge in the entry graph G leads to a latency greater than L. The entry processor P_{in} is connected to x_1 with a link of bandwidth 1, and all other links connecting P_{in} have a bandwidth $\frac{\delta}{L}$. Similarly, the link from y_{k+1} to P_{out} has a bandwidth 1, and P_{out} cannot communicate with another processor without exceeding the bound on the latency, L. Finally, the additional processors z_i are linked with a bandwidth 1 to two processors, y_i and x_{i+1}, and with a bandwidth $\frac{\delta}{L}$ to all remaining processors.

Figure 4 illustrates this platform.

• Finally we ask whether we can achieve a latency of $L = 2n^2 w^k$.

First, notice that the size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$, since both n and k need to be encoded in unary in $\mathcal{I}_1$, and the number of stages and processors in $\mathcal{I}_2$ is polynomial in n and k. The values are exponential, since the value of the parameters is bounded by w^{2k} and $w = n^3$, thus the parameters are bounded by n^{6k}. However, n^{6k} can be encoded in binary in $\mathcal{I}_2$, thus in

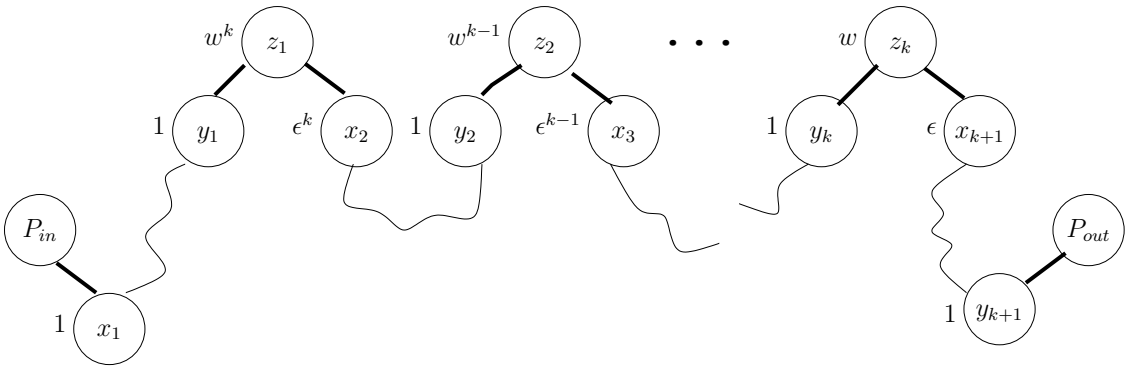


Figure 4: Target platform for the proof.

$O(k \log n)$, and it remains polynomial in the size of $\mathcal{I}_1$. The same reasoning holds for the encoding of the ϵ^k.

Suppose first that $\mathcal{I}_1$ has a solution. We derive a mapping of latency smaller than L for $\mathcal{I}_2$. We denote by ℓ_i the number of vertices in the path going from x_i to y_i in the solution of $\mathcal{I}_1$. All paths are disjoint, and they cannot include x_j or y_j with $j \neq i$, so we have $2 \leq \ell_i \leq n - 2k$, for $1 \leq i \leq k+1$, and $\sum_{i=1}^{k+1} \ell_i \leq n$.

We start by mapping the $n-2$ first stages of the pipeline on the processors of the path from x_1 to y_1, assigning one stage per processor, and all the remaining ones to processor y_1. This is possible since there are more stages than processors in the path. The computation time required to traverse these stages is $(n-2).w^k$. Then, stage of cost w^{2k} is mapped on z_1 with a computation time of $\frac{w^{2k}}{w^k} = w^k$. Finally, we map the following stage of cost ϵ^k on processor x_2, with a computation time of $\frac{\epsilon^k}{\epsilon^k} = 1$. The total computation time for stages up to the one of cost ϵ^k is thus less than $(n-2).w^k + w^k + 1 \leq n.w^k$. Only fast communication links are used, since we start by x_1 which is connected to P_{in} and we evolve through edges of the original graph. The fast communication links are also used to access z_1, entering from y_1 and exiting through x_2.

Then we keep a similar mapping, for $2 \leq i \leq k$:
- the $n-2$ first stages of cost w^{k-i+1} are mapped on the path between x_i and y_i (excluding x_i), with a total cost $(n-2).w^{k-i+1} \leq (n-2).w^k$;
- the stage of cost w^{2k-i} is mapped on processor z_i, with a cost $\frac{w^{2k-i+1}}{w^{k-i+1}} = w^k$;
- the stage of cost ϵ^{k-i+1} is mapped on processor x_{i+1}, thus achieving a cost of 1.
The total cost for these n stages is thus less than $n.w^k$.

Finally, the remaining n stages of cost 1 are mapped on the path between x_{k+1} and y_{k+1}, thus allowing to reach the output processor P_{out} with a good communication link. The cost for these stages is bounded by n.

The total number of communications does not exceed $n+k+1$, since the platform is composed of $n+k$ processors and the mapping is interval-based. Moreover, only fast communicating links (bandwidth 1) are used, and the cost of a single communication is thus $\frac{1}{n+k+1}$. Thus the cost induced by communication is bounded by 1.

The latency of this mapping is then:

$$\begin{aligned} L_{mapping} &\leq \sum_{i=1}^{k} n.w^k + \underbrace{n}_{\text{last stages}} + \underbrace{1}_{\text{communications}} \\ &\leq kn.w^k + n + 1 \\ &\leq 2n^2 w^k \leq L \end{aligned}$$

Indeed, $k \leq n$ and since $n \geq 2$, $n + 1 \leq n^2 w^k$.

The previous mapping is interval-based since we use the solution of $\mathcal{I}_1$: the paths are disjoints so we do not reuse a processor after it has been handling an interval of stages. Thus, we found a valid solution to $\mathcal{I}_2$.

Reciprocally, if $\mathcal{I}_2$ has a solution, let us show that $\mathcal{I}_1$ also has a solution. We prove that the mapping of $\mathcal{I}_2$ has to be of a similar form than the mapping described above, and thus that there exists a disjoint path between x_i and y_i, for $1 \leq i \leq k + 1$.

First let us prove that the mapping must use processor z_1 to compute the stage of cost w^{2k}. Indeed, if this stage is not processed on z_1, the best we can do is to process it on the fastest remaining processor z_2, and the corresponding cost is

$$\frac{w^{2k}}{w^{k-1}} = w^{k+1} = n^3.w^k > L$$

(we can assume $n > 2$).

Since we must use z_1 and the mapping is interval-based, z_1 must have distinct predecessor and successor processors in the mapping. However, there are only two communication links that can be used in the mapping since the latency does not exceed L. Thus, both processors y_1 and x_2 are used. The only stage that can be handled by x_2 is ϵ^k, because all other stages have a computation cost greater than ϵ^{k-1} and thus would lead to a cost $\frac{\epsilon^{k-1}}{\epsilon^k} = \frac{1}{\epsilon} = L + 1$. Therefore, since z_1 must process the stage of cost w^{2k} which is before the stage ϵ^k in the mapping, x_2 is necessarily the successor of z_1 in the mapping.

In a similar way, we prove recursively, for $i \geq 2$, that each processor z_i is used in the mapping to compute the stage of cost w^{2k-i+1}, and that x_{i+1} is its successor and it processes the stage of cost ϵ^{k-i+1}.

Let us suppose that this property is true for $j < i$. By hypothesis, $z_1, ..., z_{i-1}$ are already used to process stages preceding ϵ^{k-i+2}, and the mapping is interval-based, thus we cannot use these processors anymore. Thus, if processor z_i is not used for stage of cost w^{2k-i+1}, the best we can do is to process this stage on the fastest remaining processor, which is z_{i+1}. This leads to a cost of

$$\frac{w^{2k-i+1}}{w^{k-i}} = w^{k+1} = n^3.w^k > L$$

Therefore, z_i is used to compute this stage. Thus, the mapping must use processor x_{i+1}. The remaining stage with the lower computation cost is ϵ^{k-i+1}, since the smaller ones have already been assigned to x_j, $j < i + 1$ (by hypothesis). This one produces a cost of exactly 1, while the second smaller is ϵ^{k-i} and leads to a cost greater than L. Thus the property is true for i.

In order to ensure the latency of L, the mapping is using only fast communicating links. Thus, processors x_1 and y_{k+1} are used in the mapping. All others y_i and x_{i+1}, for $1 \leq i \leq k$, are used

because we must go through z_i. Therefore, processors must be visited in the following order:

$$x_1, y_1, z_1, x_2, ..., y_k, z_k, x_{k+1}, y_{k+1}.$$

Since it is an interval-based mapping, the processors used between x_i and y_i are all distinct, and they must be connected by edges in the graph by construction of $\mathcal{I}_2$, thus we found disjoint paths and a solution to $\mathcal{I}_1$. □

3.2.4 Non-Approximability Results

Theorem 5. *Given any constant $\lambda > 0$, there exists no λ-approximation to the* Latency-Int-Het *problem, unless $P = NP$.*

Proof. Given λ, assume that there exists a λ-approximation to the Latency-Int-Het problem. Let $\mathcal{I}_1$ be an instance of DCP (see proof of Theorem 4). We build the same instance $\mathcal{I}_2$ as in the previous proof, except for:

- The speed of the fast processors z_i, $1 \leq i \leq k$. These are set to $(\lambda w)^{k-i+1}$ instead of w^{k-i+1}.
- The speed of the slow processors x_i, $2 \leq i \leq k+1$. These are set to $\lambda^{-(k-i+2)}\epsilon^{k-i+2}$ instead of ϵ^{k-i+2}.
- The computation cost of each stage of cost w^{2k-i+1}, $1 \leq i \leq k$. Each of these costs are transformed to $(\lambda w)^{k-i+1}w^k$.
- The computation cost of each stage of cost ϵ^i, $1 \leq i \leq k$. Each of these costs are transformed to $\lambda^{-i}\epsilon^i$.

We use the λ-approximation algorithm to solve this new instance $\mathcal{I}_2$ of our problem, which returns a mapping scheme of latency L_{alg}. We thus have $L_{alg} \leq \lambda L_{opt}$, where L_{opt} is the optimal latency. Then we prove that we can solve DCP in polynomial time.

Let $L = 2n^2w^k$ be the bound on the latency used in the proof of Theorem 4.

- If $L_{alg} > \lambda L$, then $L_{opt} > L$ and there is no solution to DCP; otherwise, we could achieve a mapping of latency equal to L with a mapping similar to the one described in the proof of Theorem 4.
- If $L_{alg} \leq \lambda L$, let us prove that DCP has a solution, by ensuring that the mapping has a similar structure than in the proof of Theorem 4.
  First, we must map the stage of cost w^{2k} on processor z_1; otherwise, the best we can do is to process it on the fastest remaining processor z_2, and the corresponding cost is $\frac{(\lambda w)^k w^k}{(\lambda w)^{k-1}} = \lambda w^{k+1} > \lambda L \geq L_{alg}$. Then, we show that the only stage handled by x_2 is the stage of cost $\lambda^{-k}\epsilon^k$. Other stages have a computation cost greater than $\lambda^{-(k-1)}\epsilon^{k-1}$ and would lead to a cost $\frac{\lambda^{-(k-1)}\epsilon^{k-1}}{\lambda^{-k}\epsilon^k} = \frac{\lambda}{\epsilon} = \lambda(L+1) > \lambda L \geq L_{alg}$.
  This line of reasoning can be kept recursively, similarly as in the proof of Theorem 4, thanks to the introduction of λ in the costs. Therefore, the mapping is similar to the one described in the proof of Theorem 4, and we conclude that DCP has a solution.

Therefore, given a λ-approximation algorithm for LATENCY-INT-HET, we can answer to DCP in polynomial time, thus proving that $P = NP$. This contradicts our hypothesis and proves the non-approximability result.

□

We define by LATENCY-ONE-TO-ONE-HET the problem of finding the one-to-one mapping which minimizes latency on a heterogeneous platform. We can also prove a non-approximability result for this problem.

Theorem 6. *Given any constant $\lambda > 0$, there exists no λ-approximation to the* LATENCY-ONE-TO-ONE-HET *problem, unless $P = NP$.*

Proof. Given λ, assume that there exists a λ-approximation to the LATENCY-ONE-TO-ONE-HET problem. Consider an arbitrary instance $\mathcal{I}_1$ of the Hamiltonian Path problem HP, i.e., a complete graph $G = (V, E)$: is there an Hamiltonian path in G? This problem is known to be NP-complete [5]. We aim at showing that we can solve it in polynomial time by using the λ-approximation algorithm of LATENCY-ONE-TO-ONE-HET.

We build the following instance $\mathcal{I}_2$ of LATENCY-ONE-TO-ONE-HET: we consider an application with $n = |V|$ identical stages. All application costs are unit costs: $w_i = \delta_i = 1$ for all i. For the platform, in addition to P_{in} and P_{out} we use n identical processors of unit speed: $s_i = 1$ for all i. We simply write i for the processor P_i that corresponds to vertex $v_i \in V$. We only play with the link bandwidths: we interconnect P_{in} and P_{out} to all other processors with links of bandwidth 1. Also, if $(i, j) \in E$, then we interconnect i and j with a link of bandwidth 1. All the other links are slow: their bandwidth is set to $\frac{1}{\lambda(2n+2)}$. We ask whether we can achieve a latency not greater than $L = 2n + 1$. This transformation can clearly be done in polynomial time, and the size of $\mathcal{I}_2$ is linear in the size of $\mathcal{I}_1$.

Now we use the λ-approximation algorithm to solve this new instance $\mathcal{I}_2$ of our problem, which returns a mapping scheme of latency L_{alg}. We thus have $L_{alg} \leq \lambda L_{opt}$, where L_{opt} is the optimal latency. Then we prove that we can solve HP in polynomial time.

- If $L_{alg} > \lambda L$, then $L_{opt} > L$ and there is no solution to HP. Otherwise, let $v_1, ..., v_n$ be the hamiltonian path in G. If we map stage S_i onto processor i, for all i, then we obtain a mapping of cost $2n + 1 = L$. Indeed, the total cost for computations is n, and only edges of the original graph, of bandwidth 1, are used in the mapping, thus adding a cost of $n + 1$ for communications. This contradicts the fact that $L_{opt} > L$. Therefore, HP has no solution.

- If $L_{alg} \leq \lambda L$, let us prove that HP has a solution. The mapping is not using any slow link, otherwise it would induce a cost $\lambda(2n+2) = \lambda(L+1) > \lambda L$, which contradicts our hypothesis. Since the mapping is one-to-one, all n nodes must be used in the mapping, and it defines a hamiltonian path.

Thus, depending on the result of the algorithm, we can answer the HP problem, which proves that P=NP. This completes the proof.

□

4 Conclusion

In this paper, we have studied the problem of mapping linear chain applications onto large-scale heterogeneous platforms, focusing onto latency minimization. The latency measures the response time of the system in order to process one single data set entirely, and it is a key parameter for the user. When the platform has homogeneous communications, it is straightforward to provide the optimal mapping, be it one-to-one, interval-based or general.

When moving to fully heterogeneous platforms, the situation changes dramatically. Only the general mapping problem remains polynomial (and we provided a new proof of this result). While the one-to-one mapping problem was known to become NP-hard, that of the interval mapping problem was left open. The main result of this paper is to fill the gap and to prove its NP-hardness through quite an involved reduction. Furthermore, we prove that neither the interval mapping problem nor the one-to-one mapping problem can be approximated by any constant factor (unless P=NP). All these results constitute an important step in assessing the difficulty of the various mapping strategies that have been studied in the literature.

References

[1] Anne Benoit, Veronika Rehn-Sonigo, and Yves Robert. Optimizing latency and reliability of pipeline workflow applications. In *HCW'2008, the 17th International Heterogeneity in Computing Workshop*. IEEE Computer Society Press, 2008.

[2] P.B. Bhat, C.S. Raghavendra, and V.K. 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.

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

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

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

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

[7] 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.

[8] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and load balancing in parallel and distributed systems*. IEEE Computer Science Press, 1995.

[9] Jaspal Subhlok and Gary 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.

[10] Jaspal Subhlok and Gary 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.