Scheduling multiple divisible loads on a linear processor network

Matthieu Gallet, Yves Robert, Frédéric Vivien

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Scheduling multiple divisible loads on a linear processor network

Matthieu Gallet — Yves Robert — Frédéric Vivien

N° 6235

June 2007

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--6235--FR+ENG

Scheduling multiple divisible loads on a linear processor network

Matthieu Gallet, Yves Robert, Frédéric Vivien

Thème NUM — Systèmes numériques
Projet GRAAL

Rapport de recherche n° 6235 — June 2007 — 16 pages

Abstract: Min, Veeravalli, and Barlas have recently proposed strategies to minimize the overall execution time of one or several divisible loads on a heterogeneous linear network, using one or more installments [18, 19]. We show on a very simple example that their approach does not always produce a solution and that, when it does, the solution is often suboptimal. We also show how to find an optimal schedule for any instance, once the number of installments per load is given. Then, we formally state that any optimal schedule has an infinite number of installments under a linear cost model as the one assumed in [18, 19]. Therefore, such a cost model cannot be used to design practical multi-installment strategies. Finally, through extensive simulations we confirmed that the best solution is always produced by the linear programming approach, while solutions of [19] can be far away from the optimal.

Key-words: scheduling, heterogeneous processors, divisible loads, single-installment, multiple-installments.

This text is also available as a research report of the Laboratoire de l'Informatique du Parallélisme `http://www.ens-lyon.fr/LIP`.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Ordonnancement de tâches divisibles sur un réseau linéaire de processeurs

Résumé : Min, Veeravalli, and Barlas ont proposé [18, 19] des stratégies pour minimiser le temps d'exécution d'une ou de plusieurs tâches divisibles sur un réseau linéaire de processeurs hétérogènes, en distribuant le travail en une ou plusieurs tournées. Sur un exemple très simple nous montrons que l'approche proposée dans [19] ne produit pas toujours une solution et que, quand elle le fait, la solution est souvent sous-optimale. Nous montrons également comment trouver un ordonnancement optimal pour toute instance, quand le nombre de tournées par tâches est spécifié. Ensuuite, nous montrons formellement que lorsque les fonctions de coûts sont linéaires, comme c'est le cas dans [18, 19], un ordonnancement optimal a un nombre infini de tournées. Un tel modèle de coût ne peut donc pas être utilisé pour définir des stratégies en multi-tournées utilisables en pratique. Finalement, au moyen de simulations exhaustives, nous montrons que la meilleure solution est toujours produite par l'approche par programmation linéaire, tandis que les solutions de [19] peuvent être très éloignées de l'optimal.

Mots-clés : ordonnancement, ressources hétérogènes, tâches divisibles, tournées.

1 Introduction

Efficiently scheduling the tasks of a parallel application onto the resources of a distributed computing platform is critical for achieving high performance. This scheduling problem has been studied for a variety of application models. Some popular models consider a set of independent tasks without task synchronization nor inter-task communications. Among these models some focus on the case in which the number of tasks and the task sizes can be chosen arbitrarily. This corresponds to the case when the application consists of an amount of computation, or *load*, that can be arbitrarily divided into any number of independent pieces of arbitrary sizes. This corresponds to a perfectly parallel job: any sub-task can itself be processed in parallel, and on any number of workers. In practice, this model is an approximation of an application that consists of (very) large numbers of identical, low-granularity computations. This *divisible load* model has been widely studied in the last several years, and *Divisible Load Theory* (DLT) has been popularized by the landmark book written in 1996 by Bharadwaj, Ghose, Mani and Robertazzi [4]. DLT has been applied to a large spectrum of scientific problems, including linear algebra [6], image processing [12, 15], video and multimedia broadcasting [1, 2], database searching [5], biological pattern-matching [14], and the processing of large distributed files [17].

Divisible load theory provides a practical framework for the mapping of independent tasks onto heterogeneous platforms. From a theoretical standpoint, the success of the divisible load model is mostly due to its analytical tractability. Optimal algorithms and closed-form formulas exist for the simplest instances of the divisible load problem. We are aware of only one NP-completeness result in the DLT [20]. This is in sharp contrast with the theory of task graph scheduling, which abounds in NP-completeness theorems and in inapproximability results.

Several papers in the Divisible Load Theory field consider master-worker platforms [4, 8, 3]. However, in communication-bound situations, a linear network of processors can lead to better performance: on such a platform, several communications can take place simultaneously, thereby enabling a pipelined approach. Recently, Min, Veeravalli, and Barlas have proposed strategies to minimize the overall execution time of one or several divisible loads on a heterogeneous linear processor network [18, 19]. Initially, the authors targeted single-installment strategies, that is strategies under which a processor receives in a single communication all its share of a load. But for cases where their approach failed to design single-installment strategies, they also considered multi-installment solutions.

In this paper, we first show on a very simple example (Section 3) that the approach proposed in [19] does not always produce a solution and that, when it does, the solution is often suboptimal. The fundamental flaw of the approach of [19] is that the authors are optimizing the scheduling load by load, instead of attempting a global optimization. The load by load approach is suboptimal and unduly over-constrains the problem. On the contrary, we show (Section 4) how to find an optimal scheduling for any instance, once the number of installments per load is given. In particular, our approach always find the optimal solution in the single-installment case. We also formally state (Section 5) that under a linear cost model for communication and communication, as in [18, 19], an optimal schedule has

an infinite number of installments. Such a cost model can therefore not be used to design practical multi-installment strategies. Finally, in Section 6, we report the simulations that we performed in order to assess the actual efficiency of the different approaches. We now start by introducing the framework.

2 Problem and Notations

We use a framework similar to that of [18, 19]. The target architecture is a linear chain of m processors $(P_1, P_2, \ldots, P_m)$. Processor P_i is available from time τ_i. It is connected to processor P_{i+1} by the communication link l_i (see Figure 1). The target application is composed of N loads, which are *divisible*, which means that each load can be split into an arbitrary number of chunks of any size, and these chunks can be processed independently. All the loads are initially available on processor P_1, which processes a fraction of them and delegates (sends) the remaining fraction to P_2. In turn, P_2 executes part of the load that it receives from P_1 and sends the rest to P_3, and so on along the processor chain. Communications can be overlapped with (independent) computations, but a given processor can be active in at most a single communication at any time-step: sends and receives are serialized (this is the full *one-port* model).

Since the last processor P_m cannot start computing before having received its first message, it is useful for P_1 to distribute the loads in several installments: the idle time of remote processors in the chain will be reduced due to the fact that communications are smaller in the first steps of the overall execution.

The objective is to minimize the *makespan*, i.e., the time at which all loads are completed. For the sake of convenience, all notations are summarized in Table 1.

We deal with the general case in which the nth load is distributed in Q_n installments of different sizes. For the jth installment of load n, processor P_i takes a fraction $\gamma_j^n(i)$, and sends the remaining part to the next processor while processing its own fraction.

Loads have different characteristics: load n (with $1 \leq n \leq N$) is defined by a volume of data $V_{comm}(n)$ and a quantity of computation $V_{comp}(n)$. Moreover, processors and links are not identical either. We let w_i be the time taken by P_i to compute a unit load ($1 \leq i \leq m$), and z_i be the time taken by P_i to send a unit load to P_{i+1} (over link l_i, $1 \leq i \leq m-1$). Note that we assume a linear model for computations and communications, as in the original articles [18, 19], and as is often the case in divisible load literature [16, 9] (we will discuss this hypothesis in Section 5).

For the jth installment of the nth load, let $Comm_{i,n,j}^{start}$ denote the starting time of the communication between P_i and P_{i+1}, and let $Comm_{i,n,j}^{end}$ denote its completion time; similarly, $Comp_{i,n,j}^{start}$ denotes the start time of the computation on P_i for this installment, and $Comp_{i,n,j}^{end}$ denotes its completion time. Following [18, 19], we make the assumption that processor P_i sends the relevant fraction of the jth installment of the nth load to processor P_{i+1} *before* it starts to receive another fraction of load from P_{i-1}. Similarly, we suppose that the order in which the different application loads are sent is fixed. Although very natural,

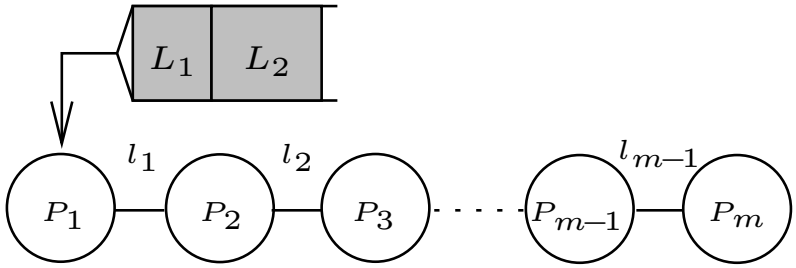

Figure 1: Linear network, with m processors and $m-1$ links.

m	Number of processors in the system.
P_i	Processor i, where $i = 1, \ldots, m$.
w_i	Time taken by processor P_i to compute a unit load.
z_i	Time taken by P_i to transmit a unit load to P_{i+1}.
τ_i	Availability date of P_i (time at which it becomes available for processing the loads).
N	Total number of loads to process in the system.
Q_n	Total number of installments for nth load.
$V_{comm}(n)$	Volume of data for nth load.
$V_{comp}(n)$	Volume of computation for nth load.
$\gamma_i^j(n)$	Fraction of nth load computed on processor P_i during the jth installment.
$Comm_{i,n,j}^{start}$	Start time of communication from processor P_i to processor P_{i+1} for jth installment of nth load.
$Comm_{i,n,j}^{end}$	End time of communication from processor P_i to processor P_{i+1} for jth installment of nth load.
$Comp_{i,n,j}^{start}$	Start time of computation on processor P_i for jth installment of nth load.
$Comp_{i,n,j}^{end}$	End time of computation on processor P_i for jth installment of nth load.

Table 1: Summary of notations.

these assumptions do reduce the solution space, and it might be useful to relax them in some special cases.

3 Motivating example

We first recall the algorithms presented in [19]. We then introduce our motivating example and use it to assess the performance of these algorithms.

3.1 The existing algorithms

It is often *stated* that, when scheduling a single load under the divisible load model, in an optimal solution "all participating processors stop computing at the same time instant" [19]. This property has been formally proved for some particular settings [3, 8] but is used far more generally and some existing proofs are even flawed (see [8] for examples).

Min, Veeravalli, and Barlas use this *optimality principle* to build their algorithm. They assume that all processors participate in the processing of each load and all complete simultaneously the processing of any given load. The strict application of this principle leads to what we call the SingleInst algorithm. In order to further optimize the processing of the loads, they force each processor to stay busy from the first time it starts processing a load to the overall completion. When such a solution does not exist with a single-installment strategy, that is when a processor receives in a single communication all its share of a given load, they resort to multi-installment strategies where each installment is the largest possible satisfying all the constraints (all processors complete simultaneously an installment processing). This defines their main algorithm, that we call MultiInst. The idea is to fully overlap communications by computations (which is obviously not always possible when communications are far more expensive than computations). Both algorithms optimize the schedule load by load, instead of attempting a global optimization.

3.2 The example

Our motivating example uses 2 identical processors P_1 and P_2 with $w_1 = w_2 = \lambda$, and $z_1 = 1$. We consider $N = 2$ identical divisible loads to process, with $V_{comm}(1) = V_{comm}(2) = 1$ and $V_{comp}(1) = V_{comp}(2) = 1$. Note that when λ is large, communications become negligible and each processor is expected to process around half of both loads. But when λ is close to 0, communications are very important, and the solution is not obvious. As both processors have the same computational power, under MultiInst they will process the same fraction of any given installment of any given load, except for the first installment of the first load.

To ease the reading, we only give a short (intuitive) description of the schedules, and we provide the different makespans without justification; all details can be found in the research report [7].

We first consider a simple schedule which uses a single installment for each load, as illustrated in Figure 2. Processor P_1 computes a fraction $\gamma_1^1(1) = \frac{2\lambda^2+1}{2\lambda^2+2\lambda+1}$ of the first load, and a fraction $\gamma_1^1(2) = \frac{2\lambda+1}{2\lambda^2+2\lambda+1}$ of the second load. Then the second processor computes a fraction $\gamma_2^1(1) = \frac{2\lambda}{2\lambda^2+2\lambda+1}$ of the first load, and a fraction $\gamma_2^1(2) = \frac{2\lambda^2}{2\lambda^2+2\lambda+1}$ of the second load. The makespan achieved by this schedule is equal to $\text{makespan}_1 = \frac{2\lambda\left(\lambda^2+\lambda+1\right)}{2\lambda^2+2\lambda+1}$.

3.3 Case $\lambda \geq \frac{\sqrt{3}+1}{2}$: single-installment

Under the algorithms of [19], P_1 and P_2 have to simultaneously complete the processing of their share of the first load. The same holds true for the second load. We are in the one-

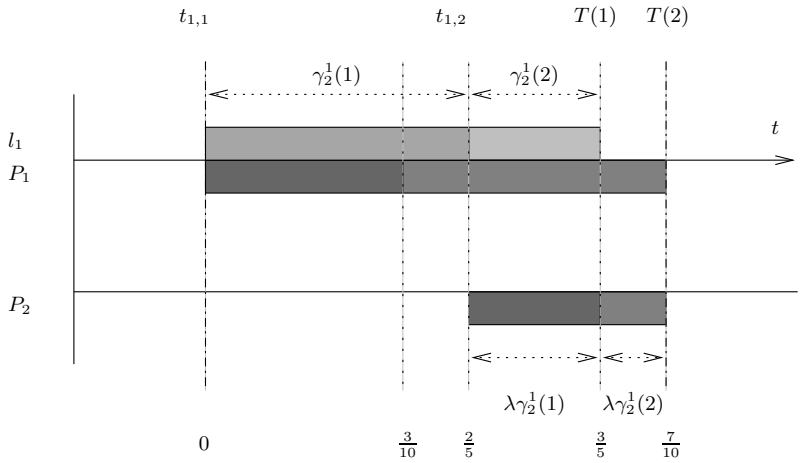

Figure 2: A possible schedule when $\lambda = \frac{1}{2}$.

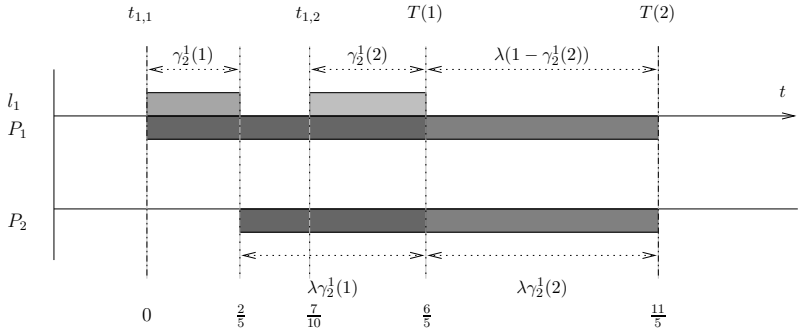

Figure 3: The schedule of [19] for $\lambda = 2$.

installment case when P_1 is fast enough to send the second load to P_2 while it is computing the first load (hence SingleInst and MultiInst have the same output). This condition writes $\lambda \geq \frac{\sqrt{3}+1}{2} \approx 1.366$. Then, P_1 processes a fraction $\gamma_1^1(1) = \frac{\lambda+1}{2\lambda+1}$ of the first load, and a fraction $\gamma_1^1(2) = \frac{1}{2}$ of the second one. The makespan achieved by this schedule is $\text{makespan}_2 = \frac{\lambda(4\lambda+3)}{2(2\lambda+1)}$.

Comparing both makespans, we have $0 \leq \text{makespan}_2 - \text{makespan}_1 \leq \frac{1}{4}$, the solution of [19] having a strictly larger makespan, except when $\lambda = \frac{\sqrt{3}+1}{2}$. A visual representation of this case is given in Figure 3 for $\lambda = 2$.

3.4 Case $\lambda < \frac{\sqrt{3}+1}{2}$: multi-installment

The solution of [19] is a multi-installment strategy when $\lambda < \frac{\sqrt{3}+1}{2}$, i.e., when communications tend to be important compared to computations. More precisely, this case happens when P_1 does not have enough time to completely send the second load to P_2 before the end of the computation of the first load on both processors.

The way to proceed in [19] is to send the second load using a multi-installment strategy. Q_2 denote the number of installments for this second load. We can easily compute the size of each fraction distributed to P_1 and P_2. Processor P_1 has to process a fraction $\gamma_1^1(1) = \frac{\lambda+1}{2\lambda+1}$ of the first load, and fractions $\gamma_1^1(2), \gamma_1^2(2), \ldots, \gamma_1^{Q_2}(2)$ of the second one. Moreover, for $1 \leq k < Q_2$, due to all the assumptions, we have $\gamma_1^k(2) = \lambda^k \gamma_2^1(1)$. And for $k = Q_2$ (the last installment), we have $\gamma_1^{Q_2}(2) \leq \lambda^{Q_2}\gamma_2^1(1)$. We can then establish an upper bound on the portion of the second load distributed in Q_2 installments:

$$\sum_{k=1}^{Q_2} \left(2\gamma_1^k(2)\right) \leq 2\sum_{k=1}^{Q_2} \left(\gamma_2^1(1)\lambda^k\right) = \frac{2\left(\lambda^{Q_2}-1\right)\lambda^2}{2\lambda^2-\lambda-1}$$

if $\lambda \neq 1$, and $Q_2 = 2$ otherwise. We have three cases to discuss:

1. $0 < \lambda < \frac{\sqrt{17}+1}{8} \approx 0.64$: Since $\lambda < 1$, we can write for any nonnegative integer Q_2:

$$\sum_{k=1}^{Q_2} \left(2\gamma_1^k(2)\right) < \sum_{k=1}^{\infty} \left(2\gamma_2^1(1)\lambda^k\right) = \frac{2\lambda^2}{(1-\lambda)(2\lambda+1)}$$

   $\frac{2\lambda^2}{(1-\lambda)(2\lambda+1)} < 1$ when $\lambda < \frac{\sqrt{17}+1}{8}$. So, an infinite number of installments do not suffice to completely process the second load. In other words, no solution is found in [19] for this case. A visual representation of this case is given in Figure 4 with $\lambda = 0.5$.

2. $\lambda = \frac{\sqrt{17}+1}{8}$: Then $\frac{2\lambda^2}{(1-\lambda)(2\lambda+1)} = 1$, and an infinite number of installments is required to completely process the second load. This solution is unrealistic.

3. $\frac{\sqrt{17}+1}{8} < \lambda < \frac{\sqrt{3}+1}{2}$: The solution of [19] is then a multi-installment solution which is better than any solution using a single installment per load. (A visual representation of this case is given in Figure 5 with $\lambda = 1$.) However this solution may require a very large number of installments. Furthermore, this solution is not optimal. Indeed, consider the case $\lambda = \frac{3}{4}$. The algorithm of [19] achieves a makespan equal to $\left(1-\gamma_2^1(1)\right)\lambda + \frac{\lambda}{2} = \frac{9}{10}$. The first load is sent in one installment and the second one is sent in 3 installments, as the number of installments is set in [19] as $Q_2 = \left\lceil \frac{\ln(\frac{4\lambda^2-\lambda-1}{2\lambda^2})}{\ln(\lambda)} \right\rceil$. However, we can come up with a better schedule by splitting both loads into two installments, and distributing them as follows:

   - Load 1, first round: P_1 processes 0 unit;

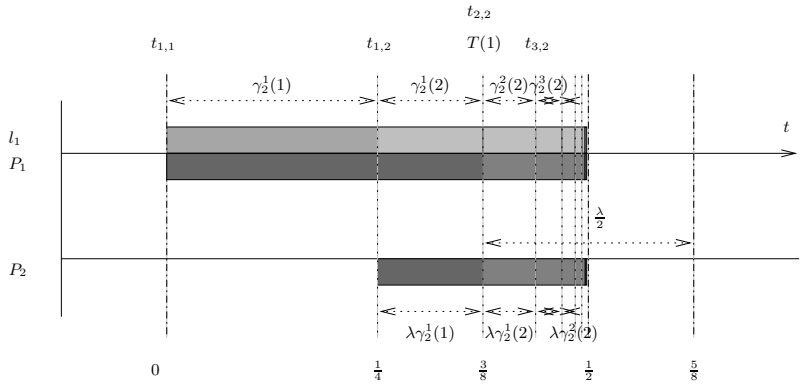

Figure 4: The schedule of [19] for $\lambda = \frac{1}{2}$.

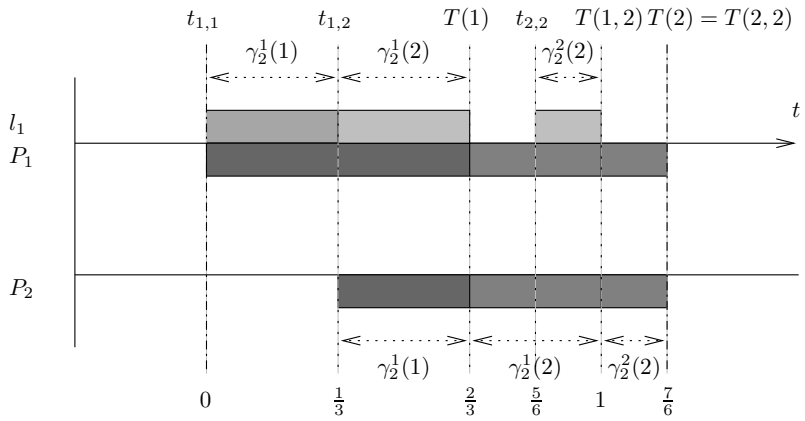

Figure 5: The schedule of [19] for $\lambda = 1$.

- Load 1, first round: P_2 processes $\frac{192}{653}$ unit;
- Load 1, second round: P_1 processes $\frac{317}{653}$ unit;
- Load 1, second round: P_2 processes $\frac{144}{653}$ unit;
- Load 2, first round: P_1 processes 0 unit;
- Load 2, first round: P_2 processes $\frac{108}{653}$ unit;
- Load 2, second round: P_1 processes $\frac{464}{653}$ unit;
- Load 2, second round: P_2 processes $\frac{81}{653}$ unit.

This scheme gives us a total makespan equal to $\frac{781}{653}\frac{3}{4} \approx 0.897$, which is (slightly) better than 0.9. This shows that among the schedules having a total number of four installments, the solution of [19] is suboptimal.

3.5 Conclusion

Despite its simplicity (two identical processors and two identical loads), out motivating example clearly outlines the limitations of the approach of [19]: this approach does not always return a feasible solution and, when it does, this solution is not always optimal. In the next section, we show how to compute an optimal schedule when dividing each load into any prescribed number of installments. Our simulations will later show that the gap between MultiInst and the optimal schedule can be significantly large.

4 Optimal solution

We now show how to compute an optimal schedule, when dividing each load into any prescribed number of installments. Therefore, when this number of installment is set to 1 for each load (i.e., $Q_n = 1$, for any n in $[1, N]$), the following approach solves the problem originally targeted by Min, Veeravalli, and Barlas.

To build our solution we use a linear programming approach. In fact, we only have to list all the (linear) constraints that must be fulfilled by a schedule, and write that we want to minimize the makespan. All these constraints are captured by the linear program in Figure 6. The optimality of the solution comes from the fact that the constraints are exactly all the constraints that any schedule must fulfill under the assumptions of Section 2, and a solution to the linear program is obviously always feasible. This linear program simply encodes the following constraints (a constraint has the same number below and in Figure 6):

1. P_i cannot start a new communication to P_{i+1} before the end of the corresponding communication from P_{i-1} to P_i,
2. P_i cannot start to receive the next installment of the nth load before having finished to send the current one to P_{i+1},
3. P_i cannot start to receive the first installment of the next load before having finished to send the last installment of the current load to P_{i+1},
4. any transfer has to begin at a nonnegative time,
5. the duration of any transfer is equal to the product of the time taken to transmit a unit load by the volume of data to transfer,
6. processor P_i cannot start to compute the jth installment of the nth load before having finished to receive the corresponding data,
7. the duration of any computation is equal to the product of the time taken to compute a unit load by the volume of the computation,
8. processor P_i cannot start to compute the first installment of the next load before it has completed the computation of the last installment of the current load,

$$
\begin{array}{rcccl}
\forall i < m-1, n \leq N, j \leq Q_n & Comm^{start}_{i+1,n,j} & \geq & Comm^{end}_{i,n,j} & (1)\\
\forall i < m-1, n \leq N, j < Q_n & Comm^{start}_{i,n,j+1} & \geq & Comm^{end}_{i+1,n,j} & (2)\\
\forall i < m-1, n < N & Comm^{start}_{i,n+1,1} & \geq & Comm^{end}_{i+1,n,Q_n} & (3)\\
\forall i \leq m-1, n \leq N, j \leq Q_n & Comm^{start}_{i,n,j} & \geq & 0 & (4)\\
\forall i \leq m-1, n \leq N, j \leq Q_n & Comm^{end}_{i,n,j} & = & Comm^{start}_{i,n,j} + z_i V_{comm}(n) \sum_{k=i+1}^{m} \gamma^j_k(n) & (5)\\
\forall i \geq 2, n \leq N, j \leq Q_n & Comp^{start}_{i,n,j} & \geq & Comm^{end}_{i,n,j} & (6)\\
\forall i \leq m, n \leq N, j \leq Q_n & Comp^{end}_{i,n,j} & = & Comp^{start}_{i,n,j} + w_i \gamma^j_i(n) V_{calc}(n) & (7)\\
\forall i \leq m, n < N & Comp^{start}_{i,n+1,1} & \geq & Comp^{end}_{i,n,Q_n} & (8)\\
\forall i \leq m, n \leq N, j < Q_n & Comp^{start}_{i,n,j+1} & \geq & Comp^{end}_{i,n,j} & (9)\\
\forall i \leq m & Comp^{start}_{i,1,1} & \geq & \tau_i & (10)\\
\forall i \leq m, n \leq N, j \leq Q_n & \gamma^j_i(n) & \geq & 0 & (11)\\
\forall n \leq N & \sum_{i=1}^{m} \sum_{j=1}^{Q} \gamma^j_i(n) & = & 1 & (12)\\
\forall i \leq m & \text{makespan} & \geq & Comp^{end}_{i,N,Q} & (13)
\end{array}
$$

Figure 6: The complete linear program.

9. processor P_i cannot start to compute the next installment of a load before it has completed the computation of the current installment of that load,
10. processor P_i cannot start to compute the first installment of the first load before its availability date,
11. the portion of a load dedicated to a processor is necessarily nonnegative,
12. any load has to be completely processed,
13. the makespan is no smaller than the completion time of the last installment of the last load on any processor.

Altogether, we have a linear program to be solved over the rationals, hence a solution in polynomial time [11]. In practice, standard packages like GLPK [10] will return the optimal solution for all reasonable problem sizes. Note that the linear program gives the optimal solution for a prescribed number of installments for each load. In the next section we discuss the problem of the number of installments.

5 Possible extensions

Several of the model restrictions can be alleviated. First the model uses *uniform machines*, meaning that the speed of a processor does not depend on the task that it executes. It is

easy to extend the linear program for unrelated parallel machines, introducing w_i^n to denote the time taken by P_i to process a unit load of type n. Also, all processors and loads are assumed to be available from the beginning. In our linear program, we have introduced availability dates for processors. The same way, we could have introduced release dates for loads. Furthermore, instead of minimizing the makespan, we could have targeted any other objective function which is an affine combination of the loads completion time and of the problem characteristics, like the average completion time, the maximum or average (weighted) flow, etc.

The formulation of the problem does not allow any piece of the n'th load to be processed before the nth load is completely processed, if $n' > n$. We can easily extend our solution to allow for N rounds of the N loads, each load being still divided into several installments. This would allow to interleave the processing of the different loads.

The divisible load model is linear, which causes major problems for multi-installment approaches. Indeed, once we have a way to find an optimal solution when the number of installments per load is given, the question is: what is the optimal number of installments? Under a linear model for communications and computations, the optimal number of installments is infinite, as the following theorem states:

Theorem 1. *Assuming a linear cost model for communications and computations, consider any problem with one or more loads and at least two processors. Then, any schedule using a finite number of installments is suboptimal for makespan minimization.*

This theorem is proved by building, from any schedule, another schedule with a strictly smaller makespan. The proof is available in the research report [7].

An infinite number of installments obviously does not define a feasible solution. Moreover, in practice, when the number of installments becomes too large, the model is inaccurate, as acknowledged in [4, pp. 224 and 276]. Any communication incurs a startup cost K, which we express in bytes. Consider the nth load, whose communication volume is $V_{comm}(n)$: it is split into Q_n installments, and each installment requires $m-1$ communications. The ratio between the actual and estimated communication costs is roughly equal to $\rho = \frac{(m-1)Q_nK+V_{comm}(n)}{V_{comm}(n)} > 1$. Since K, m, and V_{comm} are known values, we can choose Q_n such that ρ is kept relatively small, and so such that the model remains valid for the target application. Another, and more accurate solution, would be to introduce latencies in the model, as in [3]. This latter article shows how to design asymptotically optimal multi-installment strategies for star networks. A similar approach could be used for linear networks.

6 Experiments

Using simulations, we now assess the relative performance of our linear programming approach, of the solutions of [18, 19], and of simpler heuristics. We first describe the experimental protocol and then analyze the results.

Experimental protocol. We use Simgrid [13] to simulate linear processor networks. Schedules are computed by a Perl script, and their validity and theoretical makespan are checked before running them in the simulator.

We study the following algorithms and heuristics:

- The naive heuristic SIMPLE distributes each load in a single installment and proportionally to the processor speeds.
- The strategy for a single load, SINGLELOAD, presented by Min and Veeravalli in [18]. For each load, we set the time origin to the availability date of the first communication link (in order to try to prevent communication contentions).
- The SINGLEINST strategy described in Section 3.1.
- The MULTIINST n strategy. This is a slightly modified version of MULTIINST which ensures that a load is not distributed in more than n installments, the nth installment distributing all the load remaining work.
- The HEURISTIC B presented by Min, Veeravalli and Barlas in [19].
- LP n: the solution of our linear program where each load is distributed in n installments.

We measure the relative performance of each heuristic on each instance: we divide the makespan obtained by a given heuristic on a given instance by the smallest makespan obtained, on that instance, among all heuristics. Considering the relative performance enables us to obtain meaningful statistics among instances with very different makespans.

Instances. We emulate a heterogeneous linear network with $m = 10$ processors. We consider two distribution types for processing powers: *homogeneous* where each processor P_i has a processing power $\frac{1}{w_i} = 100$ MFLOPS, and *heterogeneous* where processing powers are uniformly picked between 10 and 100 MFLOPS. Communication link l_i has a speed $\frac{1}{z_i}$ uniformly chosen between 10 Mb/s and 100 Mb/s, and a latency between 0.1 and 1 ms (links with high bandwidths having small latencies). For homogeneous and heterogeneous platforms, simulation tasks have their computation volumes either all uniformly distributed between 6 GFLOPS and 4 TFLOPS, or all uniformly distributed between 6 and 60 GFLOPS. For each combination of processing power distribution and task size, we fix the communication to computation volume of all tasks to either $0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50$, or 100 (bytes per FLOPS). Each instance contains 50 loads. Finally, we randomly built 100 instances per combination of the different parameters, hence a total of 3,600 instances simulated and reported in Table 2. The code and the experimental results can be downloaded from: `http://graal.ens-lyon.fr/~mgallet/downloads/DivisibleLoadsLinearNetwork.tar.gz`.

We fixed an upper-bound for the number of installments per load used by the different heuristics: MULTIINST to either 100 or 300, SINGLELOAD to 100, and LP n to either 1, 2, 3, or 6.

Discussions of the results. We first remark that the linear program approach always reaches the best makespan. LP 1, LP 2, LP 3, and LP 6 achieve equivalent performance, always less than 0.5% away from the optimal. This may seem counter-intuitive but can be

Heuristic	Average	Std dev.	Max
Simple	1150.42887	$1.6\ 10^{3}$	8385.94163
SingleLoad 100	1462.65842	$2.0\ 10^{3}$	10714.41753
SingleInst	1.06307	$8.0\ 10^{-2}$	1.52324
MultiInst 100	1.13962	$1.8\ 10^{-1}$	1.98712
MultiInst 300	1.13963	$1.8\ 10^{-1}$	1.98712
Heuristic B	1.13268	$1.7\ 10^{-1}$	2.01865
LP 1	1.00047	$8.5\ 10^{-4}$	1.00498
LP 2	1.00005	$9.6\ 10^{-5}$	1.00196
LP 3	1.00002	$4.7\ 10^{-5}$	1.00098
LP 6	1.00000	0	1.00001

Table 2: Summary of results.

readily explained: multi-installment strategies mainly reduce the idle time incurred on each processor before it starts processing the first task, and the room for improvement is thus quite small in our (and [19]) batches of 50 tasks. The strict one-port communication model forbids the overlapping of some communications due to different installments, and further limits the room for performance enhancement. Except in some peculiar cases, distributing the loads in multi-installments do not induce significant gains. In very special cases, LP 6 does not achieve the best performance during the simulations, but this fact can be explained by the latencies existing in simulations.

The bad performance of Simple, which can have makespans 8000 greater than the optimal, justify the use of sophisticated scheduling strategies. SingleInst has tremendously better performance than SingleLoad as it far better takes into account communication link availabilities: the huge difference of performance is due to the instances with expensive communications. SingleInst achieves very good average performance, within 6% of the optimal. It also achieves significantly better performance than MultiInst, and Heuristic B. This may also be due to the fact that multi-installment strategies are not efficient in our experimental context. The slight difference performance between MultiInst 100 and MultiInst 300 shows that MultiInst sometimes uses a large amount of installments for an insignificant negative gain (certainly due to the latencies). When communication links are slow and when computations dominate communications, MultiInst and Heuristic B can have makespans 98% higher than the optimal.

7 Conclusion

We have shown that a linear programming approach allows to solve all instances of the scheduling problem addressed in [18, 19]. In contrast, the original approach was providing a solution only for particular problem instances. Moreover, the linear programming approach

returns an optimal solution for any number of installments, while the original approach was empirically limited to very special strategies, and was often sub-optimal.

Intuitively, the solution of [19] is less efficient than the schedule of Section 3.2 because it aims at locally optimizing the makespan for the first load, and then optimizing the makespan for the second one, and so on, instead of directly searching for a global optimum. We were not able to provide closed-form expressions characterizing optimal solutions, but, owing to the power of linear programming, we were able to derive an optimal schedule for any problem instance. We validated this approach through simulations which confirmed that the best solution is always produced by the linear programming approach, while solutions of [19] can be far away from the optimal. The simulations also show that, in our settings, the multi-installment strategies rarely lead to significant gains.

References

[1] D. Altilar and Y. Paker. An optimal scheduling algorithm for parallel video processing. In *IEEE Int. Conference on Multimedia Computing and Systems*, 1998.

[2] D. Altilar and Y. Paker. Optimal scheduling algorithms for communication constrained parallel processing. In *Euro-Par 2002*, LNCS 2400, pages 197–206. Springer Verlag, 2002.

[3] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang. Scheduling divisible loads on star and tree networks: results and open problems. *IEEE Trans. Parallel Distributed Systems*, 16(3):207–218, 2005.

[4] V. Bharadwaj, D. Ghose, V. Mani, and T. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, 1996.

[5] J. Blazewicz, M. Drozdowski, and M. Markiewicz. Divisible task scheduling - concept and verification. *Parallel Computing*, 25:87–98, 1999.

[6] S. Chan, V. Bharadwaj, and D. Ghose. Large matrix-vector products on distributed bus networks with communication delays using the divisible load paradigm: performance and simulation. *Mathematics and Computers in Simulation*, 58:71–92, 2001.

[7] M. Gallet, Y. Robert, and F. Vivien. Comments on "design and performance evaluation of load distribution strategies for multiple loads on heterogeneous linear daisy chain networks". Research report RR-6123, INRIA, 2007. http://hal.inria.fr/inria-00130294.

[8] S. Genaud, A. Giersch, and F. Vivien. Load-balancing scatter operations for grid computing. *Parallel Computing*, 30(8):923–946, 2004.

[9] D. Ghose and T. Robertazzi, editors. *Special issue on* Divisible Load Scheduling. Cluster Computing, 6, 1, 2003.

[10] GLPK: GNU Linear Programming Kit. http://www.gnu.org/software/glpk/.

[11] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of ACM STOC'84*, pages 302–311, 1984.

[12] C. Lee and M. Hamdi. Parallel image processing applications on a network of workstations. *Parallel Computing*, 21:137–160, 1995.

[13] A. Legrand, L.Marchal, and H. Casanova. Scheduling Distributed Applications: The SimGrid Simulation Framework. In *Proceedings of CCGrid'03*, pages 138–145, May 2003.

[14] A. Legrand, A. Su, and F. Vivien. Minimizing the stretch when scheduling flows of biological requests. In *Proceedings of SPAA '06*, pages 103–112. ACM Press, 2006.

[15] X. Li, B. Veeravalli, and C. Ko. Distributed image processing on a network of workstations. *Int. J. Computers and Applications (ACTA Press)*, 25(2):1–10, 2003.

[16] T. Robertazzi. Ten reasons to use divisible load theory. *IEEE Computer*, 36(5):63–68, 2003.

[17] R. Wang, A. Krishnamurthy, R. Martin, T. Anderson, and D. Culler. Modeling communication pipeline latency. In *Measurement and Modeling of Computer Systems (SIGMETRICS'98)*, pages 22–32. ACM Press, 1998.

[18] H. M. Wong and B. Veeravalli. Scheduling divisible loads on heterogeneous linear daisy chain networks with arbitrary processor release times. *IEEE Trans. Parallel Distributed Systems*, 15(3):273–288, 2004.

[19] H. M. Wong, B. Veeravalli, and G. Barlas. Design and performance evaluation of load distribution strategies for multiple divisible loads on heterogeneous linear daisy chain networks. *J. Parallel Distributed Computing*, 65(12):1558–1577, 2005.

[20] Y. Yang, H. Casanova, M. Drozdowski, M. Lawenda, and A. Legrand. On the complexity of multi-round divisible load scheduling. Research report RR-6096, INRIA, 2007. http://hal.inria.fr/inria-00123711.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399