Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Parallel Computing 29 (2003) 1121–1152

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Scheduling divisible workloads on heterogeneous platforms ☆

Olivier Beaumont [a,*], Arnaud Legrand [b], Yves Robert [b]

[a] *LaBRI, UMR CNRS 5800, Domaine Universitaire, 351, cours de la Libération, 33405 Talence Cedex, France*
[b] *LIP, UMR CNRS-INRIA 5668, ENS Lyon, 46, allée d'Italie, 69364 Lyon Cedex 07, France*

Received 17 January 2003; received in revised form 2 June 2003; accepted 11 June 2003

Abstract

In this paper, we discuss several algorithms for scheduling divisible workloads on heterogeneous systems. Our main contributions are (i) new optimality results for single-round algorithms and (ii) the design of an asymptotically optimal multi-round algorithm. This multi-round algorithm automatically performs resource selection, a difficult task that was previously left to the user. Because it is periodic, it is simpler to implement, and more robust to changes in the speeds of the processors and/or communication links. On the theoretical side, to the best of our knowledge, this is the first published result assessing the absolute performance of a multi-round algorithm. On the practical side, extensive simulations reveal that our multi-round algorithm outperforms existing solutions on a large variety of platforms, especially when the communication-to-computation ratio is not very high (the difficult case).

Keywords: Scheduling; Divisible tasks; Multi-round algorithms; Asymptotical optimality

1. Introduction

Scheduling computational tasks on a given set of processors is a key issue for high-performance computing. In this paper, we restrict our attention to the processing

☆ A shorter version of this paper appears in the 2003 Heterogeneous Computing Workshop, IEEE Computer Society Press.

* Corresponding author.
E-mail addresses: olivier.beaumont@labri.fr (O. Beaumont), arnaud.legrand@ens-lyon.fr(A. Legrand), yves.robert@ens-lyon.fr (Y. Robert).

0167-8191/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0167-8191(03)00095-4

of independent tasks whose size (and number) are a parameter of the scheduling algorithm. This corresponds to the divisible load model which has been widely studied in the last several years, and popularized by the landmark book written by Bharadwaj et al. [1]. A divisible job is a job that can be arbitrarily split in a linear fashion among any number of processors. This corresponds to a perfectly parallel job: any subtask can itself be processed in parallel, and on any number of processors. The applications of the divisible load model encompass a large spectrum of scientific problems, including among others Kalman filtering [2], image processing [3], video and multimedia broadcasting [4,5], database searching [6,7], and the processing of large distributed files [8] (see [1] for more examples).

On the practical side, the divisible load model provides a simple yet realistic framework to study the mapping of independent tasks on heterogeneous platforms. The granularity of the tasks can be arbitrarily chosen by the user, thereby providing a lot of flexibility in the implementation tradeoffs. On the theoretical side, 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. This is in sharp contrast with the theory of task graph scheduling, which abounds in NP completeness theorems [9,10] and in inapproximability results [11,12].

In this paper, the target computing platform is a heterogeneous master/worker platform, with p worker processes running on p processors labeled $P_1, P_2, \ldots, P_p$. The master P_0 sends out chunks to workers over a network: we can think of a star-shaped network, with the master in the center. The master uses its network connection in exclusive mode: it can communicate with a single worker at any time-step. There are different scenarios for the workers, depending whether they can compute while receiving from the master (full overlap) or not. The overlap model is widely used in the literature, because it seems closer to the actual characteristics of state-of-the-art computing resources (but we point out that our results extend to both models, with and without overlap). For each communication of size α_i between the master and a worker, say P_i, we pay a latency g_i and a linear term $\alpha_i G_i$, where G_i is the inverse of the bandwidth of the link between the master P_0 and P_i. In the original model of [1], all the latencies g_i are equal to zero, hence a linear cost model. However, latencies play an important role in current architectures [13], and more realistic models use the affine cost $g_i + \alpha_i G_i$ for a message of size α_i. Finally, note that when $g_i = g$ and $G_i = G$ for $1 \leqslant i \leqslant p$, the star network can be viewed as a bus oriented network [2].

The master processor can distribute the chunks to the workers in a single round, (also called installment in [1]), so that there will be a single communication between the master and each worker. This is the simplest situation, but surprisingly the optimal solution for a heterogeneous star network is not known, even for a linear cost model. We provide the optimal solution in Section 4, thereby extending the results of [2] for bus oriented networks to heterogeneous platforms.

For large workloads, the single round approach is not efficient, because of the idle time incurred by the last processors to receive their chunks. To minimize the makespan, i.e. the total execution time, the master will send the chunks to the workers in multiple rounds: the communications will be shorter (less latency) and pipelined, and

the workers will be able to compute the current chunk while receiving data for the next one. Deriving an efficient solution becomes a challenging problem: how many rounds should be scheduled? what is the best size of the chunks for each round? Intuitively, the size of the chunks should be small in the first rounds, so as to start all the workers as soon as possible. Then the chunk size should increase to a steady state value, to be determined so as to optimize the usage of the total available bandwidth of the network. Finally the chunk size should be decreased while reaching the end of the computation. In Chapter 10 of [1], there is no quantified value provided for the number of rounds to be used. Recently, Altilar and Paker [4,5], and Yang and Casanova [14] have introduced multi-round algorithms and analytically expressed their performance. We discuss these algorithms, and others, in Section 3, which is devoted to related work. To the best of our knowledge, no optimality result has ever been obtained for multi-round algorithms on heterogeneous platforms. The most important result of this paper is to fill this gap: in Section 5, we design a periodic multi-round algorithm and we establish its asymptotic optimality. We succeed in extending this result to arbitrary platform graphs, i.e. not just star-shaped network, but arbitrary graphs with cycles and multiple paths (see Appendix A).

The rest of the paper is organized as follows. We begin with models for computation and communication costs in Section 2. Next we review related results in Section 3. Then we deal with single-round algorithms in Section 4. We proceed to multi-round algorithms in Section 5. Because of its technical nature, the extension of the asymptotically optimal multi-round algorithms to arbitrary platforms graphs is described in the Appendix A. We provide some simulations in Section 6. Finally, we state some concluding remarks in Section 7.

2. Models

As already said, we assume a total workload W_{total} that is perfectly divisible into an arbitrary number of pieces, or chunks. Usually, it is assumed that the master itself has no processing capability, because otherwise we can add a fictitious extra worker paying no communication cost to simulate the master. There is a wide acceptance in the literature on using linear costs to model computation costs. Worker P_i will require $\alpha_i w_i$ time units to process a chunk of size α_i. However, Yang and Casanova [14] suggest to add a startup cost, or computation latency, so that the cost becomes $z_i + \alpha_i w_i$ for P_i to process a chunk of size α_i; they emphasize the importance of adding such a latency to obtain realistic results in some data-sweep applications [15]. In the following, we mainly stick to linear computational costs (except for the one-round case), but we will later mention which equations to modify to take latencies into account in the multi-round algorithms.

Modeling communication costs is more difficult, and several models have been proposed. In the original approach [1], communication costs were also assumed linear. The master would need $\alpha_i G_i$ time units to send a chunk of size α_i to P_i. While acceptable for large messages, the model becomes quite unrealistic for small messages. For instance in [1], the authors recognize that infinitely small messages would

be the best solution for multi-round algorithms with this crude model. Communication latencies g_i have been introduced by Drozdowski [6] and are now widely used: [1] the master needs $g_i + \alpha_i G_i$ to send a chunk of size α_i to worker P_i. An even more accurate model has been proposed by Rosenberg [16] and further investigated by Yang and Casanova [14]. They suggest to use the expression $g'_i + \alpha_i G_i + g''_i$, where the first latency g'_i is not overlappable, while the second latency g''_i is overlappable with the next communication. The master may send another message $g'_i + \alpha_i G_i$ time units later, while the worker cannot start computing before $g'_i + \alpha_i G_i + g''_i$ time units. The overlappable latency was introduced to model pipelined networking. Again, we restrict ourselves to non-overlappable latencies, but we will indicate how to incorporate them in the design of multi-round algorithms. Finally, note that other models [17,18] assume a fixed communication cost to dispatch chunks of any size, which seems much less realistic than adopting an affine expression with a startup and a linear term proportional to the chunk size.

Next, there is to discuss the amount of computation and communication that can be overlapped. In the model *with overlap*, each worker is capable of receiving the next chunk from the master while computing the current chunk. This corresponds to workers *equipped with a front end* in [1]. In the *no overlap* model, each worker executes communications and computations sequentially. Of course this distinction of models only applies to multi-round algorithms, because in a single-round algorithm it is impossible to overlap the communication with independent computation. When dealing with multi-round algorithms in Section 5, we will elaborate results for both models, with and without overlap.

The last question is the number of simultaneous communications that can be handled by the master. With few exceptions, the *one-port model* is assumed: the master can communicate with at most a worker at a given time-step (except may be for the short time-slice corresponding to the overlappable latency). However, as pointed out by Yang and Casanova [14], the one-port model is nicely suited to LAN network connections but a multi-port model could be used for WAN network connections.

In conclusion, we retain the following model:

(1) one-port for the master (at most one communication to a worker at any time-step);
(2) communication–computation overlap for the workers;
(3) linear computation costs $\alpha_i w_i$ (or affine $z_i + \alpha_i w_i$) for a chunk of size α_i processed by P_i, $1 \leqslant i \leqslant p$;
(4) affine communication cost $g_i + \alpha_i G_i$ to send a chunk of size α_i from P_0 to worker P_i, $1 \leqslant i \leqslant p$.

We discuss some extensions of this model when dealing with multi-round algorithms.

[1] Because there is no consensus on the notations, we borrowed the notations g_i and G_i from Wang et al. [8].

3. Related results

We divide this overview into two categories: results for single-round algorithms, and results for multi-round algorithms. We restrict ourselves to master/worker platforms, which includes bus-oriented and star-shaped networks. See [1] for results on processor trees and [6] for hypercubes.

3.1. Single-round algorithms

For single-round algorithms, the first problem is to determine in which order the chunks should be sent to the different workers. Since the master can handle only one communication at a given time step, the solution is as depicted in Fig. 1. Once the communication order has been determined, the second problem is to decide how much work should be allocated to each processor P_i. The final objective is to minimize the makespan, i.e. the total execution time.

In the case of a homogeneous (bus-oriented) platform (all G_i are equal to G), and using a linear cost model for computation (all g_i are equal to zero), Robertazzi and coworkers [2,19] have derived an optimal solution, together with closed-form expressions for the makespan T_f. This solution is surprisingly simple. Let α_i denote the fraction of workload assigned to worker P_i, where $\sum_{i=1}^{p} \alpha_i = W_{\text{total}}$, and let T_i denote the time elapsed before P_i begins its processing. Thus, $T_f = \max_i(T_i + \alpha_i w_i)$.

First, one can prove that all the processors must finish their work at the same time (i.e. $T_i + \alpha_i w_i = T_f \forall i$). Indeed, otherwise, some work could be transferred from a busy processor to an idle one in order to reduce T_f. Thus, the following system of equation holds

$$\begin{cases} T_f - T_i = \alpha_i w_i & \forall 1 \leqslant i \leqslant p \\ T_{i+1} - T_i = \alpha_{i+1} G & \forall 1 \leqslant i \leqslant p-1 \end{cases}$$

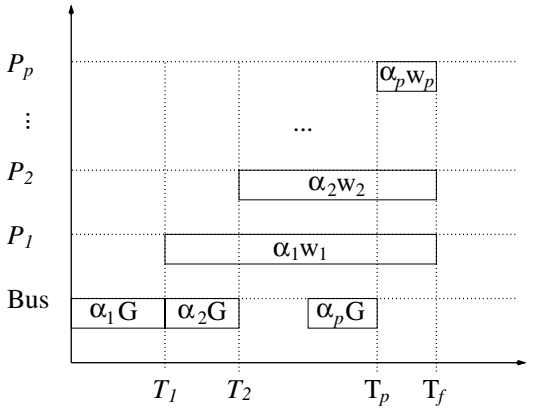

Fig. 1. Pattern of a solution for dispatching the load of a divisible job, using a bus-oriented platform ($G_i = G$). All workers complete execution at the same time-step T_f.

if data is sent successively to $P_1, \ldots, P_p$. Closed forms can be obtained for both the α_is and T_f. These closed form are rather complicated, although the method for obtaining them is elementary, and we refer the reader to [2] to find the actual algebraic expressions. The surprising and interesting point is that the overall computational time T_f does not depend upon the order chosen for sending data to the different processors, so that the ordering $P_1, \ldots, P_p$ is in fact optimal.

Later, Charcranoon et al. [20] have partially extended this work to heterogeneous (star-shaped) platforms: they still use linear communication costs, but with different G_is. The results are less satisfying than in the case of the bus. Indeed, the main known result is that if data is sent to the different processors in a given order (say, again, $P_1, \ldots, P_p$), then closed forms can be obtained for both the α_is and T_f. Unfortunately, the makespan T_f strongly depends on the communication ordering, and the result stating that all the processors must finish their work at the same time-step is no longer valid for all communication orderings. To the best of our knowledge, the optimal communication ordering is not known, and we provide the optimal solution in Section 4.

Moving to affine communication costs rather than linear communication costs, several results have been published, among others [3,6,7,16]. In 1997, Drozdowski [6] stated that the complexity of determining the optimal makespan for a general star-shaped platform (different g_is and different G_is) is not known, and to the best of our knowledge the problem is still open. We point out that Drozdowski [6] proposes an interesting mixed linear programming formulation of the problem. In the following program, $x_{i,j}$ is a boolean variable that equals 1 if P_i is chosen for the jth communication from the master

$$
\begin{aligned}
&\text{Minimize} \quad T_f \\
&\text{subject to} \\
&\begin{cases}
(1)\ \alpha_i \geqslant 0, \quad 1 \leqslant i \leqslant p \\
(2)\ \sum_{i=1}^{p} \alpha_i = W_{\text{total}} \\
(3)\ x_{i,j} \in \{0,1\}, \quad 1 \leqslant i,\ j \leqslant p \\
(4)\ \sum_{i=1}^{p} x_{i,j} = 1, \quad 1 \leqslant j \leqslant p \\
(5)\ \sum_{j=1}^{p} x_{i,j} = 1, \quad 1 \leqslant i \leqslant p \\
(6)\ \sum_{i=1}^{p} x_{1,i}(g_i + \alpha_i G_i + \alpha_i w_i) \leqslant T_f \quad \text{(first communication)} \\
(7)\ \sum_{k=1}^{j-1}\sum_{i=1}^{p} x_{k,i}(g_i + \alpha_i G_i) + \sum_{i=1}^{p} x_{j,i}(g_i + \alpha_i G_i + \alpha_i w_i) \leqslant T_f, \\
\qquad 2 \leqslant j \leqslant p \ (j\text{th communication})
\end{cases}
\end{aligned}
$$

Eq. (4) states that exactly one processor is activated for the jth communication, and Eq. (5) states that each processor is activated exactly once. Eq. (6) is a particular case

of Eq. (7), which expresses that the processor selected for the jth communication (where $j = 1$ in Eq. (6) and $j \geqslant 2$ in Eq. (7)) must wait for the previous communications to complete before its own communication and computation, and that all this quantity is a lower bound of the makespan. As pointed out by Drozdowski [6], this mixed linear program may have no solution if all the workers are not involved in the optimal solution (it may well be the case that using a strict subset of the resources proves more efficient), so the formulation is not fully general.

3.2. Multi-round algorithms

Several multi-round algorithms have been proposed in the literature [1,4,5,14] but in general they have been validated through simulations or experiments rather than with analytical formulas. This is not surprising: deriving the adequate number of rounds is a challenging task. On one hand short rounds minimize idle times in the beginning, and enable to better overlap computations and communications. On the other hand longer rounds mean less latency overheads.

Technically, a round is defined as a sequence of communications to different workers, one per worker, and deciding whether to use all workers or a strict subset of the workers is a difficult question. Even worse, should a strict subset be used, there is no reason for the subset to remain the same from one round to another.

Let $W^{(k)}$ be the total size of the chunks assigned to the workers during round k: $W^{(k)} = \sum_{i=1}^{p} \alpha_i^{(k)}$, where $\alpha_i^{(k)}$ is the chunk size of P_i at round k. Intuitively, $W^{(k)}$ should be small for the first rounds, then reach an adequate value, and then decrease in the last rounds. Yang and Casanova [14] propose that $W^{(k)}$ follows a geometric progression, and within each round that all involved processors compute for the same amount of time. [2] These simplifying assumptions enable them to derive analytical expressions for the total execution time, and the optimal number of rounds is then derived through some numerical optimization technique. The results are technically involved but very interesting. However, there remains two main limitations to this approach: (i) resource selection (determining the best subset) is performed heuristically, and (ii) there is no fundamental reason to privilege a geometric progression for the round sizes, any other monotonic and sufficiently "regular" function could be adopted.

In Section 5, we derive a periodic algorithm which is asymptotically optimal. This algorithm is simple, because rounds are repeated identically one after the others. The key-issues, i.e. the optimal number of chunks, resource selection and chunk size assignments within a chunk, are all solved through a relaxed linear program in rational numbers (hence a low-degree polynomial complexity).

[2] The geometric progression is stopped when approaching the end of the execution, so that all processors terminate working simultaneously.

4. New results for single-round algorithms

In this section, we propose a new proof method for the optimal distribution of the work to the processors in single-round algorithms. This approach enables us to retrieve some well known results, and to establish new ones.

The approach is based upon the comparison of the amount of work that is performed by the first two workers. To simplify notations, assume that P_1 and P_2 have been selected as the first two workers. There are two possible orderings, as illustrated in Fig. 2. For each ordering, we will determine the total number of tasks $\alpha_1 + \alpha_2$ that have been processed in T time units, and the total occupation t_2 of the communication medium during this time interval. We denote with upper-script (A) (resp. (B)) all the quantities related to the first (resp. second) ordering.

Let us first determine the different quantities $\alpha_1^{(A)}$, $\alpha_2^{(A)}$, $t_1^{(A)}$ and $t_2^{(A)}$ for the upper ordering in Fig. 2:

- From the equality $g_1 + z_1 + \alpha_1^{(A)}(G_1 + w_1) = T$, we deduce
$$\alpha_1^{(A)} = \frac{T - (g_1 + z_1)}{G_1 + w_1} \tag{1}$$
- Using the equality $t_1^{(A)} = g_1 + \alpha_1^{(A)} G_1$, we deduce (by Eq. (1))
$$t_1^{(A)} = \frac{g_1 w_1 + T G_1 - z_1 G_1}{G_1 + w_1} \tag{2}$$
- Using the equality $g_1 + \alpha_1^{(A)} G_1 + g_2 + z_2 + \alpha_2^{(A)}(G_2 + w_2) = T$ and Eq. (1), we deduce
$$\alpha_2^{(A)} = \frac{T - (g_2 + z_2)}{G_2 + w_2} - \frac{T G_1 + g_1 w_1 - z_1 G_1}{(G_1 + w_1)(G_2 + w_2)} \tag{3}$$
- At last, from the equality $t_2^{(A)} = g_1 + g_2 + \alpha_1^{(A)} G_1 + \alpha_2^{(A)} G_2$, we deduce (using Eqs. (1) and (3))

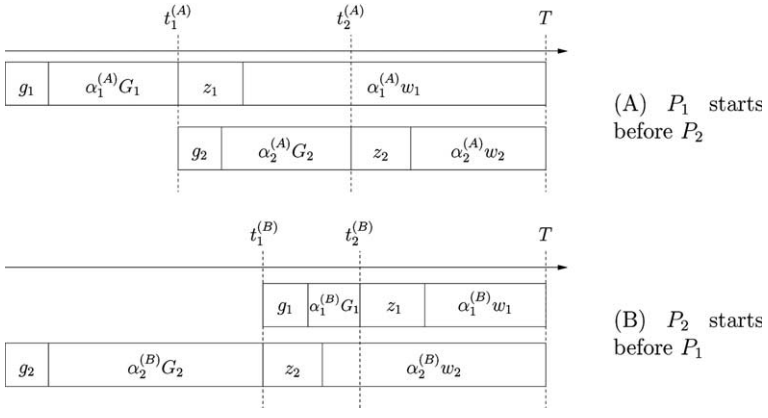

Fig. 2. Comparison of the two possible orderings.

$$t_2^{(A)} = (g_1 + g_2) + \frac{G_1(T - g_1 - z_1)}{G_1 + w_1} + \frac{G_2(T - g_2 - z_2)}{G_2 + w_2} - \frac{TG_1G_2}{(G_1 + w_1)(G_2 + w_2)} - \frac{G_2(g_1w_1 - z_1G_1)}{(G_1 + w_1)(G_2 + w_2)} \tag{4}$$

Therefore, the overall number of processed tasks is equal to (by (1) and (3))

$$\alpha_1^{(A)} + \alpha_2^{(A)} = \frac{T - (g_1 + z_1)}{G_1 + w_1} + \frac{T - (g_2 + z_2)}{G_2 + w_2} - \frac{TG_1 + g_1w_1 - z_1G_1}{(G_1 + w_1)(G_2 + w_2)}$$

and the overall occupation time of the network medium is equal (by (3)) to

$$t_2^{(A)} = (g_1 + g_2) + \frac{G_1(T - g_1 - z_1)}{G_1 + w_1} + \frac{G_2(T - g_2 - z_2)}{G_2 + w_2} - \frac{TG_1G_2}{(G_1 + w_1)(G_2 + w_2)} - \frac{G_2(g_1w_1 - z_1G_1)}{(G_1 + w_1)(G_2 + w_2)}$$

These expressions are rather complicated. Nevertheless, it is possible to obtain simple expressions when expressing the differences between situation (A) and situation (B). Indeed, we have

$$\alpha_1^{(B)} + \alpha_2^{(B)} = \frac{T - (g_1 + z_1)}{G_1 + w_1} + \frac{T - (g_2 + z_2)}{G_2 + w_2} - \frac{TG_2 + g_2w_2 - z_2G_2}{(G_1 + w_1)(G_2 + w_2)}$$

and

$$t_2^{(B)} = (g_1 + g_2) + \frac{G_1(T - g_1 - z_1)}{G_1 + w_1} + \frac{G_2(T - g_2 - z_2)}{G_2 + w_2} - \frac{TG_1G_2}{(G_1 + w_1)(G_2 + w_2)} - \frac{G_1(g_2w_2 - z_2G_2)}{(G_1 + w_1)(G_2 + w_2)}$$

Therefore, we have

$$\left(\alpha_1^{(A)} + \alpha_2^{(A)}\right) - \left(\alpha_1^{(B)} + \alpha_2^{(B)}\right) = \frac{(g_2w_2 - g_1w_1) + (z_1G_1 - z_2G_2) + T(G_2 - G_1)}{(G_1 + w_1)(G_2 + w_2)} \tag{5}$$

and

$$t_2^{(A)} - t_2^{(B)} = \frac{G_1g_2w_2 - G_2g_1w_1 + G_1G_2(z_1 - z_2)}{(G_1 + w_1)(G_2 + w_2)} \tag{6}$$

Thanks to these expressions, we can derive the optimal distribution in some special cases.

(1) No latencies $g_1 = g_2 = z_1 = z_2 = 0$, then, the occupation of the communication medium does not depend on the communication ordering, since $t_2^{(B)} = t_2^{(A)}$, by (6).

Therefore, we only need to consider the number of processed tasks in both situations. Since

$$\left(\alpha_1^{(A)} + \alpha_2^{(A)}\right) \geqslant \left(\alpha_1^{(B)} + \alpha_2^{(B)}\right) \Longleftrightarrow \text{by (5) } G_2 \geqslant G_1$$

we have better to send tasks to the processor with the smallest G_i first. This property suggests the use of a greedy algorithm, where the closest processors (in terms of high-bandwidth) are first selected. In the case of p processors, we sort them so that $G_1 \leqslant G_2 \leqslant \cdots \leqslant G_p$. We state this first result.

Theorem 1. *Consider the distribution of a divisible workload during T time units. When all communication latencies g_i and computation latencies z_i are equal to 0, sort the p processors so that $G_1 \leqslant G_2 \leqslant \cdots \leqslant G_p$ and send the tasks to the processors according to this ordering. Then, the number of processed tasks during T time units is optimal among all possible communication orderings.*

Proof. Consider an optimal ordering of the communications σ, where tasks are sent successively to $P_{\sigma(1)}, P_{\sigma(2)}, \ldots, P_{\sigma(p)}$. Let us denote by i, if it exists, the smallest index satisfying $\sigma(i) > \sigma(i+1)$. Let us consider the following ordering

$$P_{\sigma(1)}, \ldots, P_{\sigma(i-1)}, P_{\sigma(i+1)}, P_{\sigma(i)}, P_{\sigma(i+2)}, \ldots, P_{\sigma(p)}$$

Then, $P_{\sigma(1)}, \ldots, P_{\sigma(i-1)}, P_{\sigma(i+2)}, \ldots, P_{\sigma(p)}$ perform exactly the same number of tasks, since the exchange does not affect the overall communication time, but together, $P_{\sigma(i+1)}$ and $P_{\sigma(i)}$ perform $\frac{T(G_2-G_1)}{(G_1+w_1)(G_2+w_2)}$ more tasks, where T denotes the remaining time after communications to $P_{\sigma(1)}, \ldots, P_{\sigma(i-1)}$. Since $G_{\sigma(i+1)} > G_{\sigma(i)}$, there exists an optimal ordering where tasks are sent accordingly to increasing values of the G_is. □

Once the optimal ordering is known, we can use the formulas in [20] to derive the optimal assignment of works to processors, thereby filling the gap towards obtaining an optimal solution in the heterogeneous case. If all the G_is are equal, then we find the classical result of [19], stating that the number of processed tasks does not depend of the communication ordering.

(2) Optimal ordering when T is large and all the G_is are different.

We now consider the case where T is large, but both communication and computation latencies are taken into account. An interesting property is that, by (5) and (6)

$$\left(\alpha_1^{(A)} + \alpha_2^{(A)}\right) - \left(\alpha_1^{(B)} + \alpha_2^{(B)}\right) = \frac{(G_2 - G_1)}{(G_1 + w_1)(G_2 + w_2)} T + \mathrm{O}(1)$$

and

$$t_2^{(A)} - t_2^{(B)} = \mathrm{O}(1)$$

Therefore, if $G_2 > G_1$, the number of processed tasks grows linearly with T, whereas the extra communication medium occupation is bounded by a constant K, where

$$K \leqslant \frac{G_{\max}(g_{\max} w_{\max} + G_{\max} z_{\max})}{(G_{\min} + w_{\min})^2}$$

and $G_{\max}$, $g_{\max}$, $w_{\max}$, $z_{\max}$, $G_{\min}$ and $w_{\min}$ denote the maximal and minimal values of G, g, w and z over all the processors. As previously, this property suggests the use of a greedy algorithm, where the closest processors (in terms of high-bandwidth) are first selected. □

Theorem 2. *Consider the distribution of a divisible workload during T time units. Communication and computation latencies g_i and z_i, and processing speeds w_i can take arbitrary values, but we assume that the communication speeds are all dierent ($\forall i \neq j$, $G_i \neq G_j$). Sort the p processors so that $G_1 < G_2 < \cdots < G_p$ and send the tasks to the processors according to this ordering. Then, when T becomes arbitrarily large, the number of processed tasks during T time units is optimal among all possible communication orderings.*

Proof. Consider an optimal ordering of the communications σ, where tasks are sent successively to $P_{\sigma(1)}, P_{\sigma(2)}, \ldots, P_{\sigma(p)}$. Let us denote by i, if it exists, the smallest index satisfying $\sigma(i) > \sigma(i+1)$. Let us consider the following ordering

$$P_{\sigma(1)}, \ldots, P_{\sigma(i-1)}, P_{\sigma(i+1)}, P_{\sigma(i)}, P_{\sigma(i+2)}, \ldots, P_{\sigma(p)}$$

On one hand, due to the change in the ordering, the communication medium may be involved K more time units with communications to $P_{\sigma(i+1)}$ and $P_{\sigma(i)}$, where

$$K \leqslant \frac{G_{\max}(g_{\max}w_{\max} + G_{\max}z_{\max})}{(G_{\min} + w_{\min})^2}$$

The maximal number of tasks that could be processed using all the other $p-2$ processors during these extra K time units is bounded by $\frac{K(p-2)}{w_{\min}}$, i.e. $p-2$ times the number of tasks that can be processed using the fastest processor during K time units with no communication costs and no processing latencies. On the other hand, the number of extra tasks processed by $P_{\sigma(i+1)}$ and $P_{\sigma(i)}$ is of order $\frac{(G_2-G_1)}{(G_1+w_1)(G_2+w_2)}T + \mathrm{O}(1)$. Thus, when T becomes arbitrarily large, the number of extra tasks processed by $P_{\sigma(i+1)}$ and $P_{\sigma(i)}$ becomes arbitrarily large with respect to the number of tasks lost due to longer communications, what achieves the proof. □

5. Asymptotically optimal multi-round algorithms

In this section, we derive asymptotically optimal algorithms for the multi-round distribution of divisible tasks, when slave processors are either able or not to overlap their processing with incoming communications.

5.1. No overlap

The sketch of the algorithm that we propose is as follows: the overall processing time T is divided into k regular periods of duration T_p (hence $T = kT_p$, but k (and T_p) are yet to be determined).

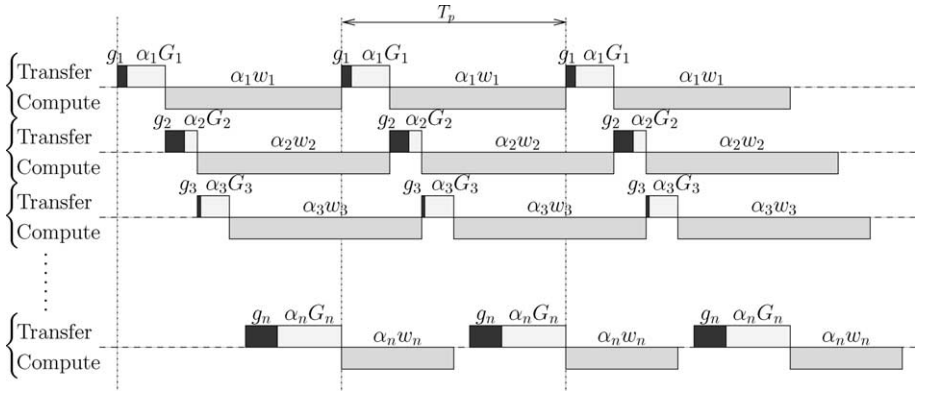

Fig. 3. Sketch of a periodic multi-round schedule using the first n workers P_1 to P_n, where $n \leqslant p$.

During a period of duration T_p, the master processor sends α_i tasks to slave processor P_i (see Fig. 3 for an example). It may well be the case that not all the processors are involved in the computation. Let $\mathscr{I} \subset \{1, \ldots, p\}$ represent the subset of indices of participating processors. For all $i \in \mathscr{I}$, the α_is must satisfy the following inequality, stating that communication resources are not exceeded

$$\sum_{i \in \mathscr{I}} (g_i + \alpha_i G_i) \leqslant T_p \tag{7}$$

Since the processors cannot overlap communications and processing, the following inequalities also hold true

$$\forall 1 \leqslant i \leqslant p, \quad i \in \mathscr{I}, \quad g_i + \alpha_i (G_i + w_i) \leqslant T_p$$

Let us denote by $\frac{\alpha_i}{T_p}$ the averaged number of tasks that slave P_i processes during one time unit, then the system becomes

$$\begin{cases} \forall 1 \leqslant i \leqslant p, \quad i \in \mathscr{I}, \quad \frac{\alpha_i}{T_p}(G_i + w_i) \leqslant 1 - \frac{g_i}{T_p} & \text{(no overlap)} \\ \sum\limits_{i \in \mathscr{I}} \frac{\alpha_i}{T_p} G_i \leqslant 1 - \frac{\sum_{i \in \mathscr{I}} g_i}{T_p} & \text{(1-port model)} \end{cases}$$

and our aim is to maximize the overall number of tasks processed during one time unit, i.e. $n = \sum_{i \in \mathscr{I}} \frac{\alpha_i}{T_p}$.

Let us consider the following linear program

$$\text{Maximize} \quad \sum_{i=1}^{p} \frac{\alpha_i}{T_p}$$

subject to

$$\begin{cases} \forall 1 \leqslant i \leqslant p, \quad \frac{\alpha_i}{T_p}(G_i + w_i) \leqslant 1 - \frac{\sum_{i=1}^{p} g_i}{T_p} \\ \sum\limits_{i=1}^{p} \frac{\alpha_i}{T_p} G_i \leqslant 1 - \frac{\sum_{i=1}^{p} g_i}{T_p} \end{cases}$$

This linear program is more constrained than previous one, since $1-\frac{g_i}{T_p}$ has been replaced by $1-\frac{\sum_{i=1}^{p} g_i}{T_p}$ in p inequalities. The linear program can be solved using a package similar to Maple [21] (we have rational numbers), but it turns out that the technique developed in [22] enables us to obtain the solution in closed form. We refer the reader to [22] for the complete proof. Let us sort the G_is so that $G_1 \leqslant G_2 \leqslant \cdots \leqslant G_p$, and let q be the largest index so that $\sum_{i=1}^{q} \frac{G_i}{G_i+w_i} \leqslant 1$. If $q < p$, let ε denote the quantity $1-\sum_{i=1}^{q} \frac{G_i}{G_i+w_i}$. If $p = q$, we set $\varepsilon = G_{q+1} = 0$, in order to keep homogeneous notations. This corresponds to the case where the full use of all the processors does not saturate the 1-port assumption for out-going communications from the master. The optimal solution to the linear program is obtained with

$$\forall 1 \leqslant i \leqslant q, \quad \frac{\alpha_i}{T_p} = \frac{1-\frac{\sum_{i=1}^{p} g_i}{T_p}}{G_i + w_i}$$

and (if $q < p$)

$$\frac{\alpha_{q+1}}{T_p} = \left(1-\frac{\sum_{i=1}^{p} g_i}{T_p}\right)\left(\frac{\varepsilon}{G_{q+1}}\right)$$

and $\alpha_{q+2} = \alpha_{q+3} = \cdots = \alpha_p = 0$.

With these values, we obtain

$$\sum_{i=1}^{p} \frac{\alpha_i}{T_p} = \left(1-\frac{\sum_{i=1}^{p} g_i}{T_p}\right)\left(\sum_{i=1}^{q} \frac{1}{G_i + w_i} + \frac{\varepsilon}{G_{p+1}}\right)$$

Let us denote by ρ_{opt} the optimal number of tasks that can be processed within one unit of time. If we denote by β_i^* the optimal number of tasks that can be processed by slave P_i within one unit of time, the β_i^*s satisfy the following set of inequalities, in which the g_is have been withdrawn

$$\begin{cases} \forall 1 \leqslant i \leqslant p, \quad \beta_i^*(G_i + w_i) \leqslant 1 \\ \sum_{i=1}^{p} \beta_i^* G_i \leqslant 1 \end{cases}$$

Here, because we have no latencies, we can safely assume that all the processors are involved (and let $\beta_i^* = 0$ for some of them). We derive that

$$\rho_{\text{opt}} \leqslant \left(1-\frac{\sum_i g_i}{T_p}\right)\left(\sum_{i}^{q} \frac{1}{G_i + w_i} + \frac{\varepsilon}{G_{q+1}}\right)$$

If we consider a large number B of tasks to be processed and if we denote by T_{opt} the optimal time necessary to process them, then

$$T_{\text{opt}} \geqslant \frac{B}{\rho_{\text{opt}}} \geqslant \frac{B}{\left(\sum_{i=1}^{q} \frac{1}{G_i+w_i} + \frac{\varepsilon}{G_{q+1}}\right)}$$

Let us denote by T the time necessary to process all B tasks with the algorithm that we propose. The number k of necessary periods satisfies $nT_pk \geqslant B$ so that we choose

$$k = \left\lceil \frac{B}{nT_p} \right\rceil$$

Therefore,

$$T \leqslant \frac{B}{n} + 2T_p \leqslant \frac{B}{\left(\sum_{i=1}^{q} \frac{1}{G_i + w_i} + \frac{\varepsilon}{G_{q+1}} \right)} \left(\frac{1}{1 - \sum_{i=1}^{p} \frac{g_i}{T_p}} \right) + T_p$$

and therefore, if $T_p \geqslant 2 \sum_{i=1}^{p} g_i$,

$$T \leqslant T_{\text{opt}} + 2 \sum_{i=1}^{p} g_i \frac{T_{\text{opt}}}{T_p} + T_p$$

Finally, if we set $T_p = \sqrt{T_{\text{opt}}}$, we check that

$$T \leqslant T_{\text{opt}} + \left(2 \sum_{i=1}^{p} g_i + 1 \right) \sqrt{T_{\text{opt}}} = T_{\text{opt}} + \mathrm{O}\left(\sqrt{T_{\text{opt}}} \right)$$

and

$$\frac{T}{T_{\text{opt}}} \leqslant 1 + \left(2 \sum_{i=1}^{p} g_i + 1 \right) \frac{1}{\sqrt{T_{\text{opt}}}} = 1 + \mathrm{O}\left(\frac{1}{\sqrt{T_{\text{opt}}}} \right)$$

which achieves of proof of the asymptotic optimality of our algorithm.

Note that resource selection is part of our explicit solution to the linear program: to give an intuitive explanation of the analytical solution, processors are greedily selected, fast-communicating processors first, as long as the communication to communication-added-to-computation ratio is not exceeded.

Also, note that it is easy to include a computation latency z_i, as suggested by Yang and Casanova [14]: simply replace g_i by $g_i + z_i$ in the formulas.

We formally state our main result.

Theorem 3. *For arbitrary values of g_i, G_i and w_i, and assuming no communication–computation overlap, the previous periodic multi-round algorithm is asymptotically optimal. Closed-form expressions for resource selection and task assignment are provided by the algorithm, whose complexity does not depend upon the total amount of work to execute.*

5.2. With overlap

In the case where slaves are able to overlap communications and processing, the algorithm that we propose in very similar to the previous one. Thus, we do not detail the proof. During time period $i + 1$, the slave processors process the tasks that they

have received during time period i, so that no processing occurs during the first period, and no communication occurs during the last period. The system of inequalities for one time unit using our algorithm becomes

$$\begin{cases} \forall 1 \leqslant i \leqslant p, \ \ i \in \mathscr{I}, \quad \dfrac{\alpha_i}{T_p} w_i \leqslant 1 & \text{(with overlap)} \\ \displaystyle\sum_{i \in \mathscr{I}} \frac{\alpha_i}{T_p} G_i \leqslant 1 - \frac{\sum_{i=1}^{p} g_i}{T_p} & \text{(1-port model)} \end{cases}$$

and we can prove, as previously that n satisfies

$$n \geqslant \left(1 - \frac{\sum_{i=1}^{p} g_i}{T_p}\right)\left(\sum_{i=1}^{q} \frac{1}{w_i} + \frac{\varepsilon}{G_{q+1}}\right)$$

where q is the largest index so that $\sum_{i=1}^{q} \frac{G_i}{w_i} \leqslant 1$ and if $q < p$, $\varepsilon = 1 - \sum_{i=1}^{q} \frac{G_i}{w_i}$. Similarly,

$$\rho_{\text{opt}} \leqslant \left(\sum_{i=1}^{q} \frac{1}{w_i} + \frac{\varepsilon}{G_{q+1}}\right)$$

and we obtain

$$T \leqslant T_{\text{opt}} + 2\left(\sum_{i=1}^{p} g_i + 1\right)\sqrt{T_{\text{opt}}}$$

and

$$\frac{T}{T_{\text{opt}}} \leqslant 1 + 2\left(\sum_{i=1}^{p} g_i + 1\right)\frac{1}{\sqrt{T_{\text{opt}}}}$$

what achieves of proof of the asymptotic optimality of our algorithm.

Theorem 4. *For arbitrary values of g_i, G_i and w_i, and assuming full communication–computation overlap, the previous periodic multi-round algorithm is asymptotically optimal. Closed-form expressions for resource selection and task assignment are provided by the algorithm, whose complexity does not depend upon the total amount of work to execute.*

5.3. Extension to arbitrary platforms graphs

We succeed in extending the design of asymptotically optimal multi-round algorithms to arbitrary platforms graphs: rather than dealing with a star-shaped network, we consider trees or even complex graphs that may involve cycles or multiple paths. These complex platforms are fully heterogeneous, both in terms of the computing speeds of the resources and in the communication capacities of the network links.

In such a complex platform, the question for the master is to decide which fraction of the workload to execute itself, and which fraction to forward to each

of its neighbors. Due to heterogeneity, the neighbors may receive different amounts of work (maybe none for some of them). Each neighbor faces in turn the same dilemma: determine which fraction of the workload to execute, and which fraction to delegate to other processors. Note that the master may well need to send tasks along multiple paths to properly feed a very fast but remote computing resource.

Due to the technical nature of the algorithms, we differ their presentation and we refer the reader to the Appendix A. In the following, we adopt a more practical point of view, and we report simulation results for star-shaped platforms.

6. Simulations

In order to evaluate our multi-round algorithm, we have crafted a simulation with the SimGrid simulator [23,24]. One major interest of relying on SimGrid is that all machine and network characteristics used in the simulations correspond to realistic values taken from the SimGrid database. We detail below the platforms that we have simulated.

In the experiments, we let the total workload size W_{total} vary in terms of workload units (or tasks) whose number range from 100 to 2000 by step of 100. Of course, the divisible load model applies here, so we assign fractional numbers of units to the processors. We let the size of a workload unit itself (i.e. the number of floating-point operations performed per unit) vary from one set of experiments to the other, so as to investigate different communication-to-computation ratios for a given application/platform pair.

In the experiments, no overlap of communications by computations was possible. We have compared our no-overlap multi-round algorithm with the multi-installment algorithm proposed in [1]. We have used a total of eleven heuristics. Three heuristics are different variants of the linear programming formulation, and the multi-installment algorithm has been tested for 1–8 installments. Here is a description of the 11 heuristics:

L.P. with fixed period. We use here the simplest variant of linear programming. We arbitrarily fix the value of the period to $T_p = 2000$. While there remains tasks to process, we allocate them to the workers according to a variant of Eq. (7): we maximize $\sum_{i=1}^{p} \alpha_i$ subject to $\sum_{i=1}^{p} (g_i + \alpha_i G_i) \leqslant T_p$ and $g_i + \alpha_i(G_i + w_i) \leqslant T_p$ for all i, $1 \leqslant i \leqslant p$. The problem is slightly over-constrained, in that we include all p latencies in Eq. (7) governing the bandwidth utilization, rather than only those of the participating processors $i \in \mathcal{I}$, as in the original formulation. Of course if the linear program returns $\alpha_i = 0$ for some i, we do not schedule the empty communication. This approximation is very good for large values of T_p, and provides a simple yet efficient task allocation if the period T_p is known a priori.

L.P. with fixed square-root period. At the beginning of the computation, an evaluation of the optimal time T needed to process the whole set of task is computed, by neglecting all latencies. This works as follows: we assume a perfectly load-bal-

ance of the work and write $(G_i + w_i)\alpha_i = \text{constant}$, with $\sum_{i=1}^{p} \alpha_i = W_{\text{total}}$. Once we have T, we let $T_p = \sqrt{T}$, and use the formulas given in Section 5.1.

L.P. with adaptive period. This is a slight modification of the previous heuristic. At each round, the period T_p is recomputed as $T_p = \sqrt{T}$, where T now is an estimation of the total time needed to process the remaining work units (rather than the total time for all units). In the very last steps of the heuristic, we stop the process and do not decrease T_p below the time needed to process the last work unit.

M.I.x. This is the multi-installment procedure of [1] with x rounds. A set of linear equations, whose number of variables depends on x, is proposed in [1]. From these equations, it is possible to derive the amount of work units to distribute to each process at each round. Note that these equations do not take in account the latency. For simulations, we use $x = 1, 2, \ldots, 8$.

6.1. Homogeneous platforms, no latency

The first set of experiments deals with homogeneous platforms, made up with PIII 1 GHz processors (delivering 114.444 Mflops), interconnected through an Ethernet 100 Mbits/s, but measured at 32.10 Mbits/s bandwidth. The measured bandwidth may seem low. These values (and all other bandwidth measures given in this paper) have been obtained during the day on a loaded network using ENV [25]. Moreover the bandwidth test are conducted using ssh (to enable bouncing on private or firewalled networks) and not raw sockets, which induces a performance loss. These values are therefore representative of what one can effectively and easily get with a robust communication technique and a correct setup of ssh keys.

The number of processors ranges from 1 to 20. One workload unit amounts to 1 GFlops of computations and 2 Mbits of data exchange. In other words, the workers communicate during 0.06 seconds to be able to compute during 9 s. The communication-to-computation ratio is quite low, which makes it easier to obtain good performances.

Fig. 4 depicts the behavior of the eleven heuristics for a 5-processor platform. The general behavior is the same for other platform sizes. We see that L.P. with fixed period and M.I.1, the one-round strategy, perform very badly. All the other strategies look similar. To outline the differences, we plot the performance ratio of a subset of the remaining heuristics over that of the L.P. with adaptive period: in Fig. 5, we use a 5-processor platform. For the sake of clarity, only M.I.x with $x = 2$ to $x = 5$ are reported in the figure; no improvement is obtained with larger values of x. The following observations can be made:

- the adaptive strategy is always very close to the best heuristic;
- the number of rounds of the multi-installment algorithm must be at least 3, and no real improvement is to be expected when increasing that number;
- the linear programming with fixed square-root period is not very regular but stays within 5% of the optimal heuristic, as soon as the number of tasks (work units) grows sufficiently.

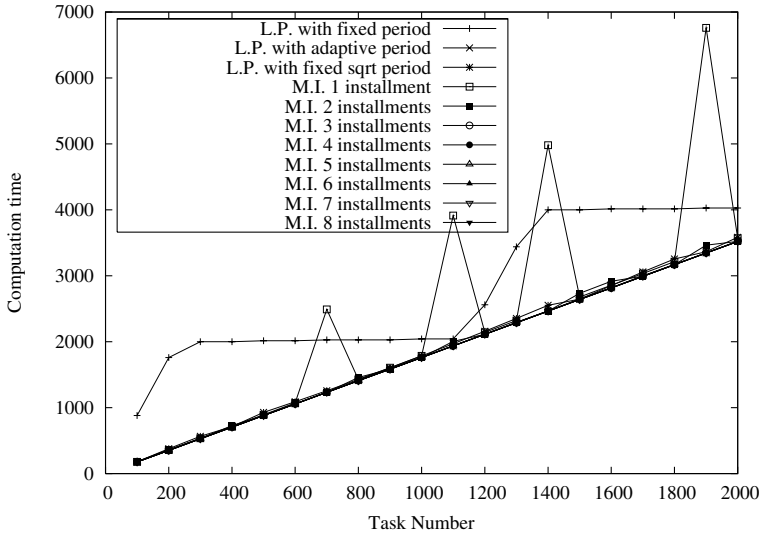

Fig. 4. Simulation time for an homogeneous platform with 5 processors.

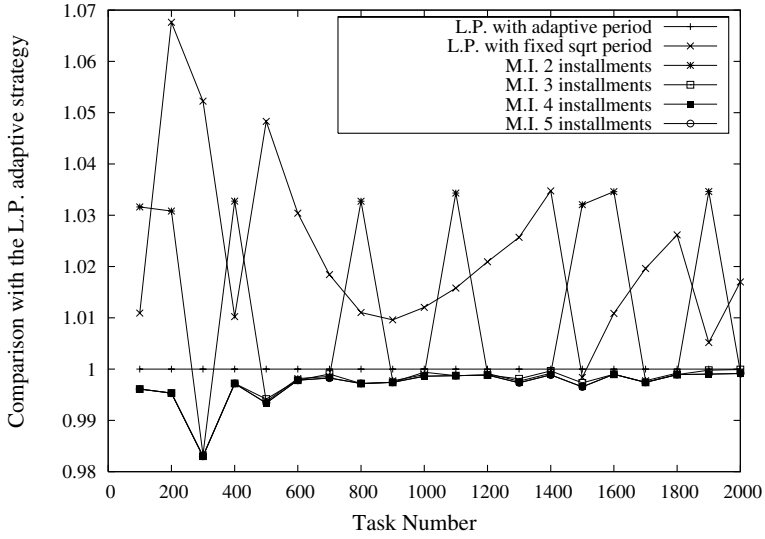

Fig. 5. Comparison with the adaptive heuristic for an homogeneous platform with 5 processors.

The behavior is similar when increasing the size of the platform. The adaptive strategy may need a larger task set to achieve the same efficiency. The number of installments needed to obtain good results with the multi-installment algorithm also becomes higher. As an example, Fig. 6 depicts the results for a 20-processor platform.

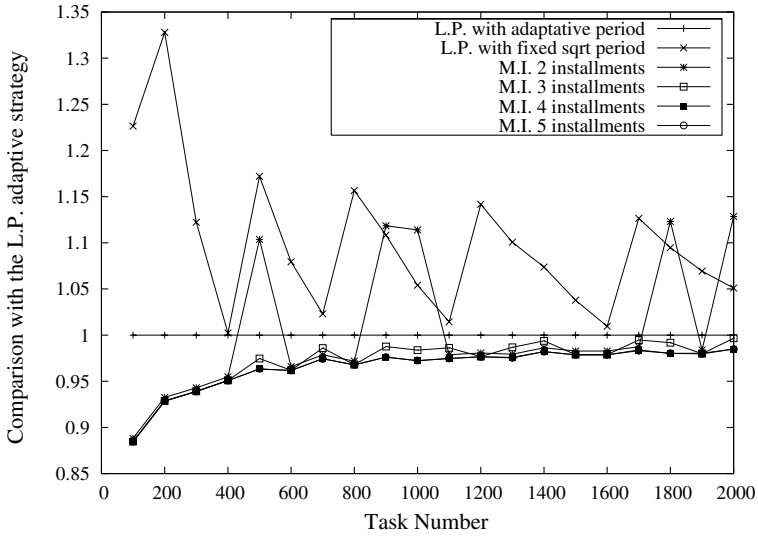

Fig. 6. Comparison with the adaptive approach for an homogeneous platform with 20 processors.

6.2. Heterogeneous platforms, no latency

Experiments have been conducted with 2000 simulated platforms made up of machines randomly chosen in the following processor set: PPro 200 MHz (22.151 Mflops), PII 450 MHz (48.492 Mflops), PII 350 MHz (34.333 Mflops), and PIII 1 GHz (114.444 Mflops). The network used to interconnect the slaves to the master could be Ethernet either 10 Mbits/s or 100 Mbits/s (we measured 4.70, 32.10 or 30.25 Mbits/s of effective bandwidth). The number of workload units ranges from 100 to 2000; with a step of 100. As before, one task unit amounts to 1 GFlops of computation and 2 Mbits of data. In other words, the workers communicate during between 0.06 and 0.4 s to be able to compute during between 9 and 90 s. Again, the communication-to-computation ratio is quite low, which makes it easier to obtain good performances.

We have run one simulation per platform and per task set, which amounts to 40,000 experiments per heuristic; the figures below correspond to averaged values. Fig. 7 and its zoomed counterpart Fig. 8 depict the comparison of the heuristics with the adaptive linear programming approach on 5-processor platforms. The following observations can be made:

- linear programming with fixed period, and one installment strategies lead to very poor performances;
- the optimal number of rounds of the multi-installment algorithm is close to 4 (for the sake of clarity, only M.I.x with $x = 2$ to $x = 5$ installments are depicted on Fig. 8 because increasing the number of rounds does not improve the performances);

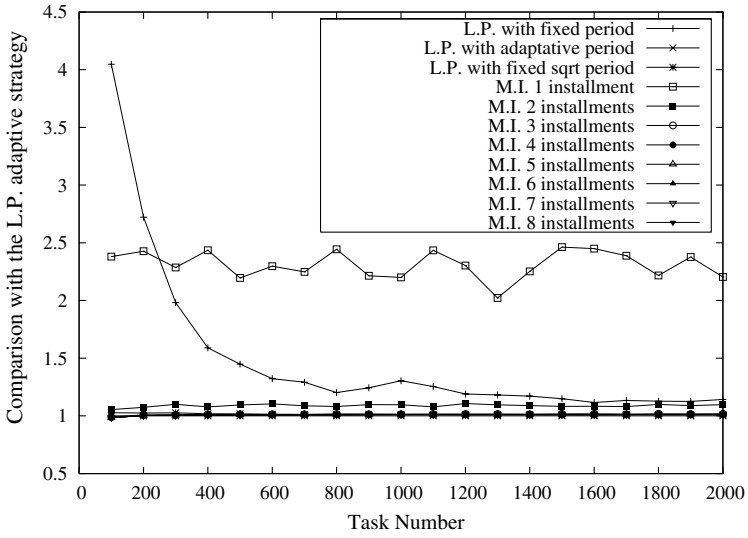

Fig. 7. Comparison with the adaptive approach for heterogeneous platforms with 5 processors.

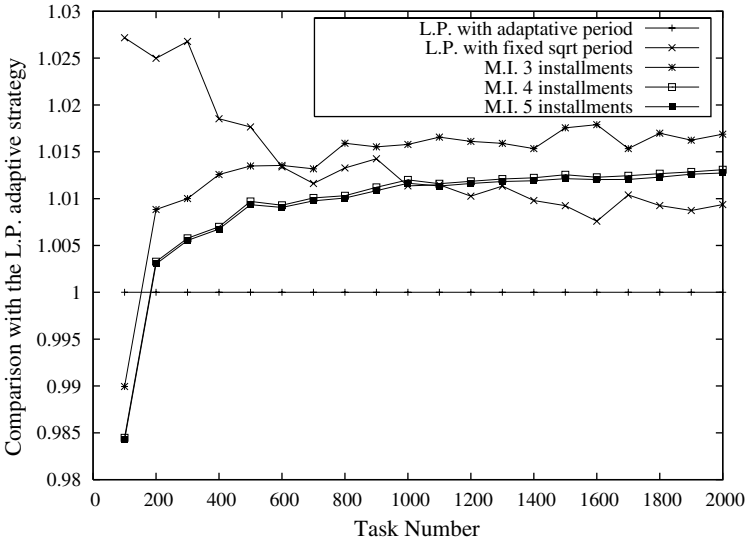

Fig. 8. Comparison with the adaptive approach for heterogeneous platforms with 5 processors (zoom).

- the linear programming with fixed square-root period slightly outperforms the multi-installment algorithm;
- the adaptive approach leads to the best performances but with a small improvement over the other good heuristics (1% in average).

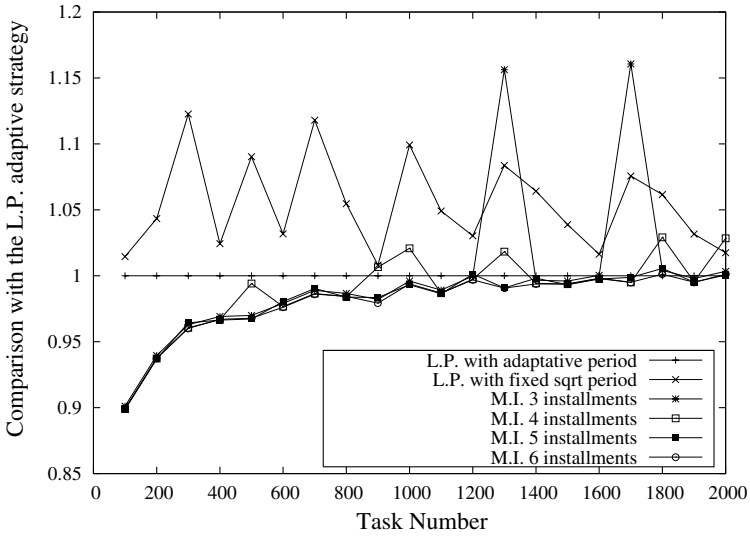

Fig. 9. Comparison with the adaptive approach for heterogeneous platforms with 20 processors.

When increasing the size of the platform, the optimal number of rounds of the multi-installment gets larger. Linear programming strategies show good performances only when the task set is large enough; the adaptive method remains better than other linear programming approaches. As an example, Fig. 9 depicts the results for 20-processor platforms.

6.3. Heterogeneous platforms, with latency

6.3.1. With a low communication-to-computation ratio

Experiments have been conducted with the same platform set and the same workloads as in Section 6.2. The only difference resides in the fact that we add 2 Mbits of data to each message, so as to model the latencies: in other words, we set $g_i = 2\ 10^6 \cdot G_i$ for every worker P_i.

Fig. 10 and its zoomed counterpart Fig. 11 depict the comparison of the heuristics with the adaptive linear programming approach on 5-processor platforms. Multi-installments with a small number of rounds still lead to poor performances. The optimal number of rounds for the multi-installment algorithm is equal to 4. The best strategy is the adaptive approach, which still leads to a small improvement of 1%. When the size of the platform gets larger (see Fig. 12), the optimal number of rounds for the multi-installment algorithm is equal to 5 and the adaptive strategy is at most 3% far away from the best other solutions (when the number of task is larger than 200).

6.3.2. With a high communication-to-computation ratio

Experiments have been conducted with, again, the same platform set as in Section 6.2, the same workloads and the same latencies. But we change the

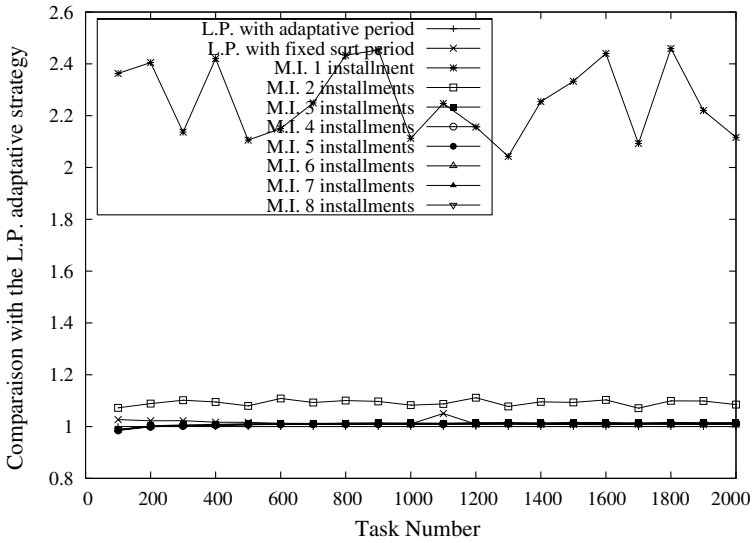

Fig. 10. Comparison with the adaptive approach for heterogeneous platforms with 5 processors, with latencies.

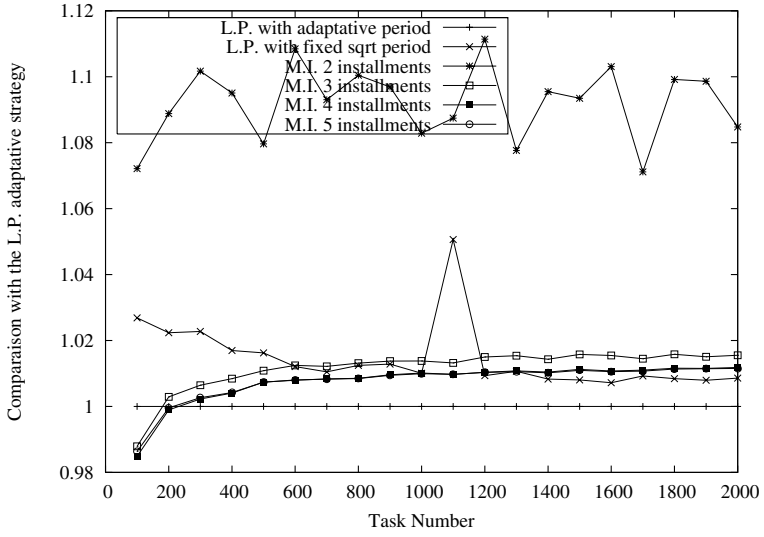

Fig. 11. Comparison with the adaptive approach for heterogeneous platforms with 5 processors, with latencies (zoom).

communication-to-computation ratio: one task unit now amounts to 50 MFlops of computation and, still, 2 Mbits of data: the communication-to-computation ratio has roughly been multiplied by 20.

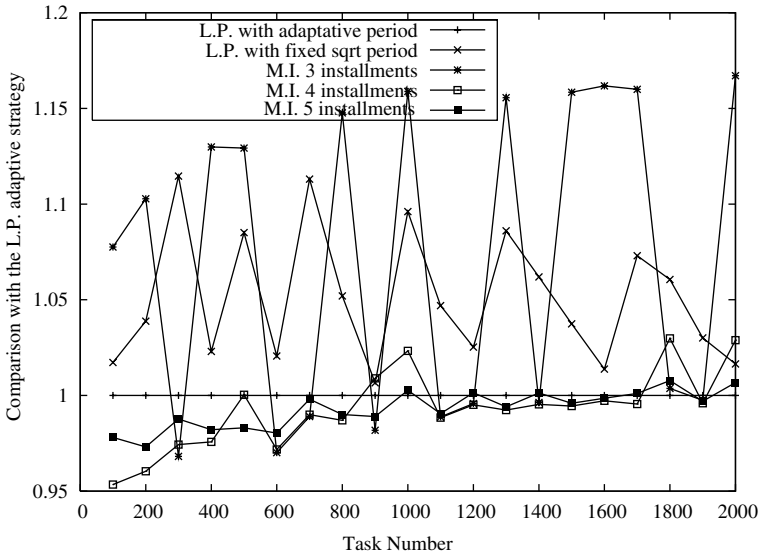

Fig. 12. Comparison with the adaptive approach for heterogeneous platforms with 20 processors, with latencies.

Figs. 13 and 14 depict the comparison of the heuristics with the adaptive linear programming approach on 5-processor and 20-processor platforms. Linear programming approaches clearly outperform any multi-installment heuristic. This is due to many reasons:

- The linear equations given in [1] cannot take latencies into account. Since the communication-to-computation ratio is getting higher, considering them becomes crucial.
- No resource selection is done in [1]. When the size of the platform gets larger, the network becomes a bottleneck and we must decide which computing resources to use. This choice is automatically performed when solving the linear inequalities.

6.4. Summary

As a general conclusion, we see that L.P. strategies, either with fixed square-root period, or with the adaptive strategy to compute the next period, are to be recommended. In most cases, they are very close to the best multi-installment solution (and determining the optimal number of rounds for this class of heuristics is a non-trivial problem). When the communication-to-computation ratio gets higher, both L.P. strategies are much better than the other heuristics.

The simulation of any of the heuristics presented in this paper on a platform made of 10 slaves takes between 0.5 s (when simulating 100 tasks) and 3.4 s (when simulating the slowest heuristic with 1000 tasks) on a Pentium III 1 GHz. It is therefore realistic to run a simulation to decide which heuristic to use.

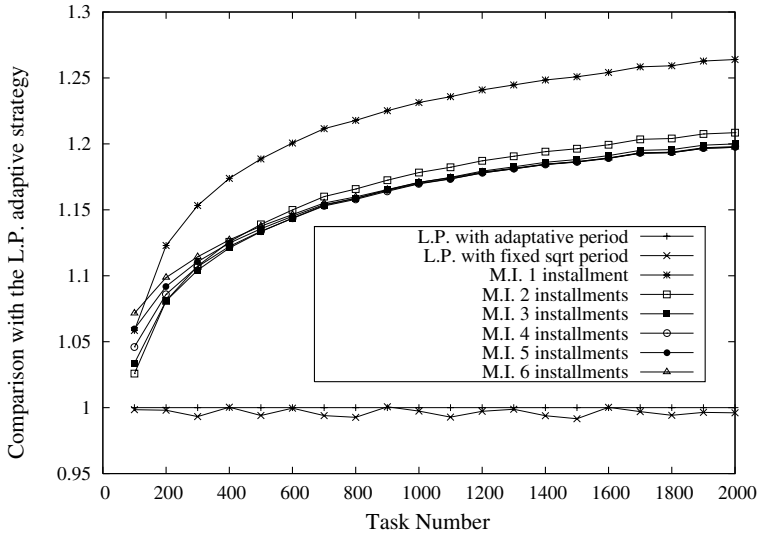

Fig. 13. Comparison with the adaptive approach for heterogeneous platforms with 5 processors, with latencies.

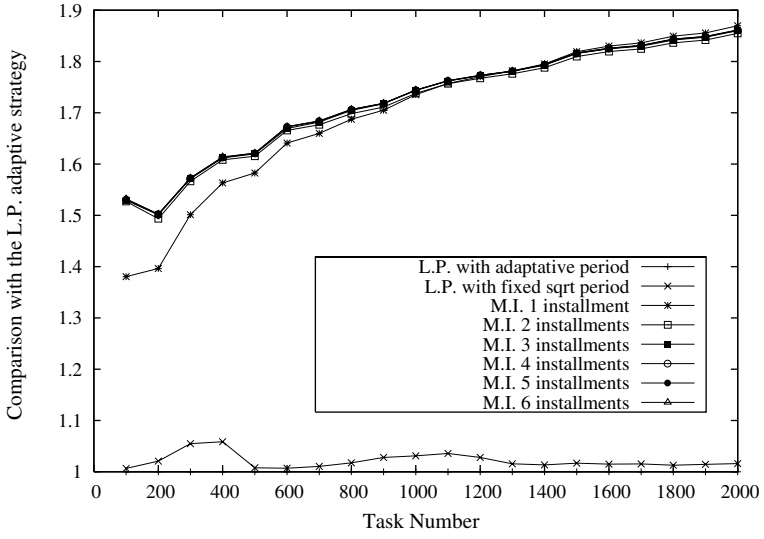

Fig. 14. Comparison with the adaptive approach for heterogeneous platforms with 20 processors, with latencies.

If we were to make a final choice, we would recommend the adaptive strategy, because it would be the most robust to changes in computation speeds or network bandwidths.

7. Conclusion

On the theoretical side, the main result of this paper is the proof of the asymptotic optimality of our multi-round algorithm. This is the first quantitative result ever assessed for a multi-round algorithm. But (maybe more importantly), our algorithm exhibits a lot of interesting features that make it a candidate of choice in a wide variety of situations:

- The best selection of the resources to be used among all available machines is automatically conducted through the linear program. Even better, resources are sorted according to the G_i and greedily selected until the sum of the ratios $\frac{G_i}{G_i+w_i}$ (without overlap) or $\frac{G_i}{w_i}$ (with overlap) exceeds 1. Previous approaches had to resort to un-guaranteed experimental heuristics.
- The best number of rounds is easily determined as a function of the task number, so there is no need to test several solutions with different round numbers, and then to select the best one.
- Because it is periodic, the algorithm is simpler to implement that other schemes that grow the chunk size repeatedly.
- For the same reason, our algorithm is more robust: the decisions taken for each round (how many work units should be sent to each worker) can be questioned before each round, thus allowing a dynamic approach to cope with, and respond to variations in computation speeds or network bandwidths. Such changes are very likely to occur, especially when the overall processing time is large. Other algorithms that rely on very long rounds in the end cannot rapidly adapt to speed or bandwidth changes.

Extensive simulations have shown that our multi-round algorithm does perform very well in practice, and significantly outperforms other heuristics when the communication-to-computation ratio of the application is not too low on the target platform. This opens up a larger range of applications for the divisible workload paradigm.

Appendix A. Asymptotically optimal multi-round algorithms on arbitrary tree platforms

In this section, we show that it is possible to derive asymptotically optimal algorithm for distributing divisible tasks onto a complex platform organized as a tree.

A.1. Platform model

The platform is represented by a tree $T = (V, E, w_i, g_{i,j}, G_{i,j})$. The nodes of the graph represent computing resources, weighted by w_i, i.e. the required time for node V_i to process an elementary task. The edges of the graph represent the interconnection network, each edge being weighted by its latency $g_{i,j}$ and the inverse of its

bandwidth $G_{i,j}$, so that the communication time for α elementary tasks along the edge $e_{i,j} = (V_i, V_j)$ of the platform takes $g_{i,j} + \alpha G_{i,j}$ time units.

A.2. Upper bound for the processing power of the platform

Our aim is to find an upper bound for the number of tasks that can be processed during T time units by the platform. To do this, we first consider an "ideal" platform, where all latencies have been removed. Clearly, the processing power of such an ideal platform is larger than the power of the actual platform. Let us now consider a node V_i during one time unit, at steady state and let us denote by α_i^{ideal} the number of tasks it processes and by $c_{i,j}^{\text{ideal}}$ the number of tasks it sends to its neighbor V_j. The number of tasks it receives from its parent V_j is therefore given by $c_{j,i}^{\text{ideal}}$. We refer to the children as the set of processors such as $P_i \to P_j$. Note that even if each processor receives data from only one other processor (remember we are working on a tree), we always refer to this processor as belonging to the set of processors such as $P_j \to P_i$. We suppose that a processor is able to overlap communications with processing and that it has one port for incoming communications and one port for outgoing communications, so that it is able at the same to receive one task from one of its neighbors (though only one), to send one task to one of its neighbor (though only one) and to process one task. This assumptions concerning the communications capabilities of the nodes will be discussed in Section A.5. The following set of inequalities states the constraints for processing (1), incoming (2) and outgoing (3) communications, and conservation law (4) for node V_i during one time step

$$\begin{cases} (1)\ \alpha_i^{\text{ideal}} w_i \leqslant 1 \\ (2)\ \displaystyle\sum_{P_j \to P_i} c_{j,i}^{\text{ideal}} G_{j,i} \leqslant 1 \quad \text{(no latencies)} \\ (3)\ \displaystyle\sum_{P_i \to P_j} c_{i,j}^{\text{ideal}} G_{i,j} \leqslant 1 \quad \text{(no latencies)} \\ (4)\ \displaystyle\sum_{P_j \to P_i} c_{j,i}^{\text{ideal}} = \alpha_i^{\text{ideal}} + \sum_{P_i \to P_j} c_{i,j}^{\text{ideal}} \end{cases}$$

Thus, at steady state, the optimal number of tasks that can be processed by the whole platform during one time unit is given by the solution of the following linear program

$$\text{Maximize} \quad \sum \alpha_i^{\text{ideal}}$$

subject to

$$\begin{cases} (1)\ \forall V_i \in V, \quad \alpha_i^{\text{ideal}} w_i \leqslant 1 \\ (2)\ \forall V_i \in V, \quad \displaystyle\sum_{P_j \to P_i} c_{j,i}^{\text{ideal}} G_{j,i} \leqslant 1 \\ (3)\ \forall V_i \in V, \quad \displaystyle\sum_{P_i \to P_j} c_{i,j}^{\text{ideal}} G_{i,j} \leqslant 1 \\ (4)\ \forall V_i \in V, \quad \displaystyle\sum_{P_j \to P_i} c_{j,i}^{\text{ideal}} = \alpha_i^{\text{ideal}} + \sum_{P_i \to P_j} c_{i,j}^{\text{ideal}} \\ (5)\ \forall V_i \in V, \quad \alpha_i^{\text{ideal}}, c_{i,j}^{\text{ideal}}, c_{j,i}^{\text{ideal}} \geqslant 0 \end{cases}$$

Let us denote by $\rho_{\text{opt}}^{\text{ideal}} = \sum_{V_i \in V} \alpha_i^{\text{ideal}}$ the optimal number of tasks processed by the "ideal" platform during one time unit. Clearly, the time necessary to process N tasks with the ideal platform (and therefore by the actual one) is upper bounded by $T_{\text{opt}} = \frac{N}{\rho_{\text{opt}}^{\text{ideal}}}$.

In the following sections, we will show that it is possible to derive an algorithm that performs N tasks with the actual platform in time of order $T_{\text{opt}} + \text{O}(\sqrt{T_{\text{opt}}})$, thus in asymptotically optimal time when N, and therefore T_{opt}, becomes arbitrarily large.

A.3. Lower bound for the processing power of the platform

Let us consider a time period of duration T_p and the processing capabilities of the actual platform. During this time period, with the assumptions concerning the overlapping and communication capabilities of the platform stated in Section A.2, let us denote by β_i the number of tasks processed by node V_i, $C_{i,j}$ (resp. $C_{j,i}$) the number of tasks sent (resp. received) by processor V_i to (resp. from) processor V_j. Then, the following set of inequalities must be satisfied

$$\begin{cases} (1)\ \beta_i w_i \leqslant T_p \\ (2)\ \displaystyle\sum_{P_j \to P_i} (C_{j,i} G_{j,i} + g_{j,i}) \leqslant T_p \quad \text{(with latencies)} \\ (3)\ \displaystyle\sum_{P_i \to P_j} (C_{i,j} G_{i,j} + g_{i,j}) \leqslant T_p \quad \text{(with latencies)} \\ (4)\ \displaystyle\sum_{P_j \to P_i} C_{j,i} = \beta_i + \sum_{P_i \to P_j} C_{i,j} \end{cases}$$

If we scale those inequalities by a factor T_p, in order to determine the processing capabilities of the actual platform during one time unit, we obtain

$$\begin{cases} (1)\ \frac{\beta_i}{T_p} w_i \leqslant 1 \\ (2)\ \displaystyle\sum_{P_j \to P_i} \frac{C_{j,i}}{T_p} G_{i,j} \leqslant 1 - \sum_{P_j \to P_i} \frac{g_{j,i}}{T_p} \quad \text{(with latencies)} \\ (3)\ \displaystyle\sum_{P_i \to P_j} \frac{C_{i,j}}{T_p} G_{i,j} \leqslant 1 - \sum_{P_i \to P_j} \frac{g_{i,j}}{T_p} \quad \text{(with latencies)} \\ (4)\ \displaystyle\sum_{P_j \to P_i} \frac{C_{j,i}}{T_p} = \frac{\beta_i}{T_p} + \sum_{P_i \to P_j} \frac{C_{i,j}}{T_p} \end{cases}$$

Let us denote by $C = 1 - \sum_{(V_k,V_l) \in E} \frac{g_{k,l}}{T_p} = 1 - \frac{A_1}{T_p}$, where $A_1 = \sum_{(V_k,V_l) \in E} g_{k,l}$. Clearly,

$$\forall i, \qquad C \leqslant 1 - \sum_{P_i \to P_j} \frac{g_{i,j}}{T_p} \quad \text{and} \quad C \leqslant 1 - \sum_{P_j \to P_i} \frac{g_{j,i}}{T_p}$$

and therefore, the solution of the following linear program provides a lower bound for the overall processing capabilities of the actual platform

$$\text{Maximize} \quad \sum \alpha_i$$

subject to

$$\begin{cases} (1)\ \forall V_i \in V, & \alpha_i w_i \leqslant C \\ (2)\ \forall V_i \in V, & \sum_{P_j \to P_i} c_{j,i} G_{j,i} \leqslant C \\ (3)\ \forall V_i \in V, & \sum_{P_i \to P_j} c_{i,j} G_{i,j} \leqslant C \\ (4)\ \forall V_i \in V, & \sum_{P_j \to P_i} c_{j,i} = \alpha_i^{\text{ideal}} + \sum_{P_i \to P_j} c_{i,j} \\ (5)\ \forall V_i \in V, & \alpha_i, c_{i,j}, c_{j,i} \geqslant 0 \end{cases}$$

Therefore, at steady state, the actual platform is able to process more than $T_p \rho_{\text{opt}}^{\text{ideal}} C$ tasks during a period of duration T_p, since the solution of the linear program for the actual platform can be obtained from solutions obtained by the linear program for the ideal platform by applying a scaling factor C (remember $C \leqslant 1$ when T_p is large enough).

In the next section, we derive from previous results an asymptotically optimal algorithm for distributing divisible tasks onto an heterogeneous complex platform.

A.4. Asymptotically optimal algorithm

The algorithm we propose consists in three main phases: an initialization phase to reach steady state, a certain number of steady state steps of duration T_p, and a clean up phase to process remaining tasks.

A.4.1. Initialization phase

During this phase, no tasks are processed, but all the processors of the platform receive the number of tasks they normally receive during each time period of duration T_p at steady state.

During such a time period at steady state, node V_i receives

$$\sum_{P_j \to P_i} C_{j,i} \leqslant \frac{T_p C}{\min_{P_j \to P_i} G_{j,i}} \leqslant \frac{1}{\min_{P_j \to P_i} G_{j,i}} T_p \leqslant \frac{1}{\min_{(V_k,V_l) \in E} G_{k,l}} T_p$$

Let us recall that the platform consists in n processors. Thus, the time necessary to send sequentially the tasks from the master to node V_i is bounded by

$$n \left(\max_{(V_k,V_l) \in E} g_{k,l} + \frac{\max_{(V_k,V_l) \in E} G_{k,l}}{\min_{(V_k,V_l) \in E} G_{k,l}} T_p \right)$$

since n is an upper bound of the length of a path from the master to V_i. Thus, the overall sequential time to send the tasks to all the processors is bounded by

$$n^2 \left(\max_{(V_k,V_l) \in E} g_{k,l} + \frac{\max_{(V_k,V_l) \in E} G_{k,l}}{\min_{(V_k,V_l) \in E} G_{k,l}} T_p \right)$$

Therefore, the time of the initialization phase is bounded by

$$A_2 + T_p A_3$$

where both $A_2 = n^2 \max g_{k,l}$ and $A_3 = n^2 \frac{\max G_{k,l}}{\min G_{k,l}}$ only depend on the platform, and neither on the overall number of tasks to be processed nor on the time period T_p.

A.4.2. Steady state phase

Since the tasks are present at each node of the platform at the end of the initialization phase, we can start the steady state phase. During step i, each processor processes the tasks it received during phase $i-1$. After k steps, the number of processed tasks is given by $kT_p\rho_{\text{opt}}^{\text{ideal}}C$. Thus, we perform k steps during the steady state phase, where

$$k = \left\lfloor \frac{N}{T_p \rho_{\text{opt}}^{\text{ideal}} C} \right\rfloor = \left\lfloor \frac{T_{\text{opt}}}{T_p C} \right\rfloor \leqslant \frac{T_{\text{opt}}}{T_p - A_1}$$

Therefore, the time of the steady state phase is bounded by

$$T_{\text{opt}} \frac{T_p}{T_p - A_1}$$

where A_1, defined in Section A.3, only depends on the platform, and neither on the overall number of tasks to be processed nor on the time period T_p.

A.4.3. Clean up phase

At the end of the kth step of the steady state phase, at most $T_p\rho_{\text{opt}}^{\text{ideal}}C \leqslant T_p\rho_{\text{opt}}^{\text{ideal}}$ tasks have not yet been processed. The time necessary to process sequentially all these tasks if all were located at the slowest processor is bounded by $T_p\rho_{\text{opt}}^{\text{ideal}} \max_i w_i$. Thus, the time of the clean up phase is bounded by

$$A_4 T_p$$

where $A_4 = \rho_{\text{opt}}^{\text{ideal}} \max_i w_i$ only depends on the platform, and neither on the overall number of tasks to be processed nor on the time period T_p.

A.4.4. Asymptotic optimality

Using the three phases algorithm we have just described, the overall processing time T^{actual} for processing N tasks on the actual platform is bounded by

$$T^{\text{actual}} \leqslant A_2 + (A_3 + A_4)T_p + T_{\text{opt}} \frac{T_p}{T_p - A_1}$$

Let us consider a time period $T_p = \sqrt{T_{\text{opt}}}$. Then, when N, and therefore T_{opt} and T_p are sufficiently large, then

$$\frac{T^{\text{actual}}}{T_{\text{opt}}} \leqslant 1 + \frac{A_3 + A_4 + 2A_1}{\sqrt{T_{\text{opt}}}} + \text{o}\left(\frac{1}{\sqrt{T_{\text{opt}}}}\right)$$

what achieves the proof of the asymptotic optimality of the platform.

A.5. Discussion of the overlapping and communication capabilities

In this section, we prove that asymptotically optimal algorithms can be derived with different assumptions concerning the overlapping capabilities of the platform. We will not provide the proofs for all possible set of assumptions, because of the length of the proofs, and we will only show the first set of inequalities used in order to compute an upper bound for the computing power of the "ideal" platforms. We recall that, if the platform is able to process simultaneously one incoming communication and one outgoing communication and is also able to overlap its processing with communications, then the set of inequalities derived in Section A.2 is the following

$$\begin{cases} (1)\ \alpha_i^{\text{ideal}} w_i \leqslant 1 \\ (2)\ \sum_{P_j \to P_i} c_{j,i}^{\text{ideal}} G_{j,i} \leqslant 1 & \text{(no latencies)} \\ (3)\ \sum_{P_i \to P_j} c_{i,j}^{\text{ideal}} G_{i,j} \leqslant 1 & \text{(no latencies)} \\ (4)\ \sum_{P_j \to P_i} c_{j,i}^{\text{ideal}} = \alpha_i^{\text{ideal}} + \sum_{P_i \to P_j} c_{i,j}^{\text{ideal}} & \text{(conservation law)} \end{cases}$$

A.5.1. Number of incoming and outgoing ports for communication

In this section, we consider that the nodes of the platforms are still able to overlap communications and processing, and we discuss the set of inequalities with respect to the number of incoming and outgoing ports.

- If there is a single port for both incoming and outgoing communications, then node V_i needs to perform all its communication using this port, and the set of inequalities becomes

$$\begin{cases} (1)\ \alpha_i^{\text{ideal}} w_i \leqslant 1 \\ (2)\ \sum_{P_j \to P_i} c_{j,i}^{\text{ideal}} G_{j,i} + \sum_{P_i \to P_j} c_{i,j}^{\text{ideal}} G_{i,j} \leqslant 1 & \text{(no latencies)} \\ (4)\ \sum_{P_j \to P_i} c_{j,i}^{\text{ideal}} = \alpha_i^{\text{ideal}} + \sum_{P_i \to P_j} c_{i,j}^{\text{ideal}} & \text{(conservation law)} \end{cases}$$

- On the other hand, if node V_i is able to perform any number of communications at the same time, then all the communications can be handled in parallel, and the set of inequalities becomes

$$\begin{cases} (1)\ \alpha_i^{\text{ideal}} w_i \leqslant 1 \\ (2)\ \forall V_j \text{ s.a. } P_j \to P_i, c_{j,i}^{\text{ideal}} G_{j,i} \leqslant 1 & \text{(no latencies)} \\ (3)\ \forall V_j \text{ s.a. } P_i \to P_j, c_{i,j}^{\text{ideal}} G_{i,j} \leqslant 1 & \text{(no latencies)} \\ (4)\ \sum_{P_j \to P_i} c_{j,i}^{\text{ideal}} = \alpha_i^{\text{ideal}} + \sum_{P_i \to P_j} c_{i,j}^{\text{ideal}} & \text{(conservation law)} \end{cases}$$

A.5.2. Overlapping capabilities

We discuss the evolution of the set of inequalities with respect to overlapping capabilities of node V_i.

- If there is a single port for both incoming and outgoing communications, then node V_i can either send, receive or process a task at a given time, and the set of inequalities becomes

$$\begin{cases} (1)\ \alpha_i^{\text{ideal}} w_i + \sum_{P_j \to P_i} c_{j,i}^{\text{ideal}} G_{j,i} + \sum_{P_i \to P_j} c_{i,j}^{\text{ideal}} G_{i,j} \leqslant 1 & \text{(no latencies)} \\ (4)\ \sum_{P_j \to P_i} c_{j,i}^{\text{ideal}} = \alpha_i^{\text{ideal}} + \sum_{P_i \to P_j} c_{i,j}^{\text{ideal}} & \text{(conservation law)} \end{cases}$$

- On the other hand, if node V_i is able to perform any number of communications at the same time, then all the communications can be handled in parallel, but processing interferes with communications, and the set of inequalities becomes

$$\begin{cases} (1)\ \forall V_j \text{ s.a. } P_j \to P_i, \alpha_i^{\text{ideal}} w_i + c_{j,i}^{\text{ideal}} G_{j,i} \leqslant 1 & \text{(no latencies)} \\ (2)\ \forall V_j \text{ s.a. } P_i \to P_j, \alpha_i^{\text{ideal}} w_i + c_{i,j}^{\text{ideal}} G_{i,j} \leqslant 1 & \text{(no latencies)} \\ (4)\ \sum_{P_j \to P_i} c_{j,i}^{\text{ideal}} = \alpha_i^{\text{ideal}} + \sum_{P_i \to P_j} c_{i,j}^{\text{ideal}} & \text{(conservation law)} \end{cases}$$

In fact, the overlapping and communication capabilities of the nodes of the platform do not need to be uniform. Since the solution are obtained with equations stating the steady state behavior of the application at each node, we can use different set of inequalities for each node, according to its respective capabilities. Therefore, we are able to derive asymptotically optimal results for a fully heterogeneous platform, where the processing power of each node, the bandwidth and the latency of each link, the number of ports for incoming and outgoing communication, and the capability of overlapping communications and processing may differ from one node to another.

References

[1] V. Bharadwaj, D. Ghose, V. Mani, T. Robertazzi, Scheduling Divisible Loads in Parallel and Distributed Systems, IEEE Computer Society Press, 1996.
[2] J. Sohn, T. Robertazzi, S. Luryi, Optimizing computing costs using divisible load analysis, IEEE Transactions on parallel and distributed systems 9 (3) (1998) 225–234.
[3] C. Lee, M. Hamdi, Parallel image processing applications on a network of workstations, Parallel Computing 21 (1995) 137–160.
[4] D. Altilar, Y. Paker, An optimal scheduling algorithm for parallel video processing, in: IEEE Int. Conference on Multimedia Computing and Systems, IEEE Computer Society Press, 1998.
[5] D. Altilar, Y. Paker, Optimal scheduling algorithms for communication constrained parallel processing, in: Euro-Par 2002, LNCS 2400, Springer Verlag, 2002, pp. 197–206.
[6] M. Drozdowski, Selected problems of scheduling tasks in multiprocessor computing systems, Ph.D. Thesis, Instytut Informatyki Politechnika Poznanska, Poznan, 1997.
[7] J. Blazewicz, M. Drozdowski, M. Markiewicz, Divisible task scheduling—concept and verification, Parallel Computing 25 (1999) 87–98.
[8] R. Wang, A. Krishnamurthy, R. Martin, T. Anderson, D. Culler, Modeling communication pipeline latency, in: Measurement and Modeling of Computer Systems (SIGMETRICS'98), ACM Press, 1998, pp. 22–32.

[9] M.R. Garey, D.S. Johnson, Computers and Intractability, a Guide to the Theory of NP-Completeness, W.H. Freeman and Company, 1991.
[10] H. El-Rewini, T.G. Lewis, H.H. Ali, Task Scheduling in Parallel and Distributed Systems, Prentice Hall, 1994.
[11] P. Chrétienne, E.G. Coffman Jr., J.K. Lenstra, Z. Liu (Eds.), Scheduling Theory and its Applications, John Wiley and Sons, 1995.
[12] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, Complexity and Approximation, Springer, Berlin, Germany, 1999.
[13] D.E. Culler, J.P. Singh, Parallel Computer Architecture: A Hardware/Software Approach, Morgan Kaufmann, San Francisco, CA, 1999.
[14] Y. Yang, H. Casanova, Multi-round algorithm for scheduling divisible workload applications: analysis and experimental evaluation, Tech. Rep. CS2002-0721, Department of Computer Science and Engineering, University of California, San Diego, 2002.
[15] H. Casanova, F. Berman, Grid'2002, in: F. Berman, G. Fox, T. Hey (Eds.), Parameter Sweeps on the Grid with APST, Wiley, 2002.
[16] A.L. Rosenberg, Sharing partitionable workloads in heterogeneous NOWs: greedier is not better, in: D.S. Katz, T. Sterling, M. Baker, L. Bergman, M. Paprzycki, R. Buyya (Eds.), Cluster Computing 2001, IEEE Computer Society Press, 2001, pp. 124–131.
[17] S.F. Hummel, E. Schonberg, L. Flynn, Factoring: a method for scheduling parallel loops, Communications of the ACM 35 (8) (1992) 90–101.
[18] T. Hagerup, Allocating independent tasks to parallel processors: an experimental study, Journal of Parallel and Distributed Computing 47 (2) (1997) 185–197.
[19] S. Bataineh, T. Hsiung, T.G. Robertazzi, Closed form solutions for bus and tree networks of processors load sharing a divisible job, IEEE Transactions on Computers 43 (10) (1994) 1184–1196.
[20] S. Charcranoon, T. Robertazzi, S. Luryi, Optimizing computing costs using divisible load analysis, IEEE Transactions on Computers 49 (9) (2000) 987–991.
[21] B.W. Char, K.O. Geddes, G.H. Gonnet, M.B. Monagan, S.M. Watt, Maple Reference Manual, 1988.
[22] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, Y. Robert, Bandwidth-centric allocation of independent tasks on heterogeneous platforms, in: International Parallel and Distributed Processing Symposium IPDPS'2002, IEEE Computer Society Press, 2002, extended version available as LIP Research Report 2001-25.
[23] H. Casanova, Simgrid: a toolkit for the simulation of application scheduling, in: Proceedings of the IEEE Symposium on Cluster Computing and the Grid (CCGrid'01), IEEE Computer Society, 2001.
[24] J. Lerouge, A. Legrand, Towards realistic scheduling simulation of distributed applications, Tech. Rep. 2002-28, LIP, ENS Lyon, France, July 2002.
[25] G. Shao, F. Berman, R. Wolski, Using effective network views to promote distributed application performance, in: International Conference on Parallel and Distributed Processing Techniques and Applications, CSREA Press, 1999.