Matthieu Gallet

Introduction to scheduling

CRC PRESS
Boca Raton London New York Washington, D.C.

Chapter 1

Divisible Load Scheduling

1.1 Introduction and motivations

1.1.1 Introduction

In this section, we focus our attention on the problem of scheduling lots of independent tasks on a parallel platform, typically a master-worker platform. We assume that we can arbitrarily split the total amount of work in parts of different sizes and distribute them to an arbitrary number of workers. The job has to be perfectly parallel, without any dependence between two sub-tasks. In practice, this model is a rational relaxation of an application made of a large number of identical, fine-grain parallel computations. This happens in many cases, like the processing of satellite pictures or the study of seismic events. This model is known as the *divisible load* model and has been widespread by Bharadwaj, Ghose, Mani and Robertazzi in [**?**] ten years ago.

In this chapter, we will focus our attention on a specific example of application and platform, namely the Earth seismic tomography using a star-shaped platform of processors. This application is used to validate a model for the internal structure of Earth (as shown in Figure 1.1), by comparing for every seismic event the propagation time of seismic waves as computed by the model with the time measured by physical devices. Each event is independent from other ones and there are large number of such events: 817,101 events were recorded only for the year 1999. The master processor owns all item, reads them and scatter them among P active workers. Then each work can process the data received from the master independently. Our objective is to minimize to total completion time, also known as the makespan. The simplified code of the program can be written as in Table 1.1.

This application respect all limitations allowing the use of the Divisible Load Theory, since it is made of a very large number of fine grain computations, and these computations are independent. We do not have any dependency, synchronization nor communication between two tasks. Many applications respect these constraints TODO donner d'autres exemples applications...

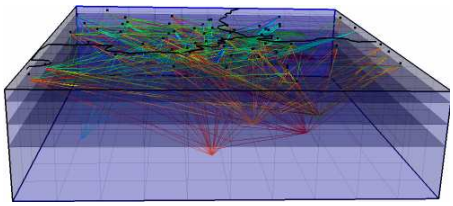

Figure 1.1: Model for the internal structure of Earth.

```
if rank == MASTER then
   raydata ← Read n lines from data file;
   MPI_Scatter(raydata, n/P, ..., rbuff, ...,
               MASTER, MPI_COMM_WORLD);
end if
Compute_work(rbuff)
```

Table 1.1: Simplified code for assessing the model quality

We wil consider two different types of platforms. The first one is a bus-shaped master-worker platform, where all workers are connected to the master through the same link, and the second one is a star-shaped master-worker platform, where workers are connected to the master through links with different characteristics.

In the next subsection 1.1.2, we will first solve our problem in a classical fashion, without using the Divisble Load Theory. Then in subsection 1.1.3 we will use the DLT to find the solution.

1.1.2 Classical approach

In this subsection, we will not use the Divisible Load Theory and we will solve our problem in a classical fashion.

The targeted platform is the bus-shaped network, as shown in Figure 1.2. Workers are *a priori* heterogeneous, and thus they have different computation speeds. According to the bus-shaped network, they share the same link to the master, and we can represent the bus-shaped network as a standard star-shaped network with homogeneous communication links.

We will use the following notations, illustrated by Figure **??**:

- M is the master processor, which initially holds all events to process.
- We have p workers, denoted as $P_1, \ldots, P_p$. In order to simplify some equations, P_0 will be the master processor M.
- Worker P_i takes a time w_i to execute a unit-size load. Of course, M takes a time w_0 to process such a load.

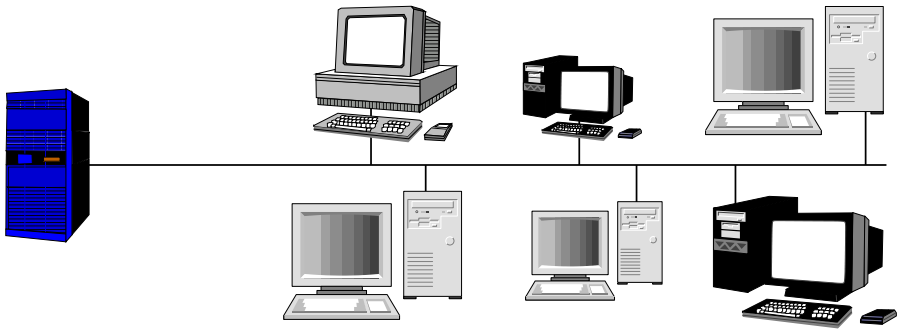

Figure 1.2: Example of bus-shaped network.

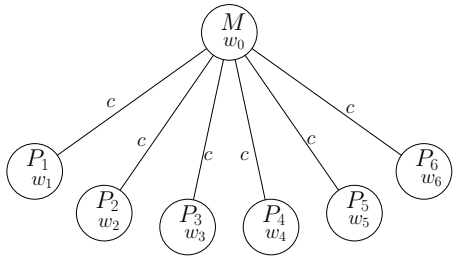

Figure 1.3: Theoritical model of a bus-shaped network.

- Any worker P_i needs c time units to receive a unit-size load from the master. We recall that all worker communicate at the same speed with the master, as specified by the bus-shaped network.
- M initially holds a total number of tasks equal to W_{total}. In practice, W_{total} is a very large integer.
- M will allocate n_i tasks to worker P_i. n_i is a integer, and since all tasks have to be processed, we have $\sum_{i=0}^{p} n_i = W_{total}$ (we consider that the master can process several tasks).
- By definition, the computation time C_i of processor P_i needed for procesing its n_i tasks is equal to $n_i.w_i$.
- The completion time T_i corresponds to the end of the computation of P_i. It is also equal to the sum of the waiting time of P_i (since P_i is idle while previous processors are communicating with M), the communication time of P_i with M and the computation time itself.

We allow the overlap of communications by computations on the master, i.e., the master can send data to workers while computing its own data.

Nonetheless, workers cannot begin their computations before having received all their data. We respect the full 1-port communication model: the master can communicate to at most one worker at a time. Finally, any worker receives all its data in one message.

As we said before, we want to minimize the total completion time T, needed to compute all the W_{total} tasks. Then we have to determine the values for the n_is, which minimize T.

Let us compute the completion time T_i of processor P_i: Since we assume that processors are served in the order $P_1, \ldots, P_p$, equations are simple:

- P_0: $T_0 = n_0.w_0$
- P_1: $T_1 = n_1.c + n_1.w_1$
- P_2: $T_2 = n_1.c + n_2.c + n_2.w_2$
- P_3: $T_2 = n_1.c + n_2.c + n_3.c + n_3.w_3$
  $\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$
- P_i: $T_i = \sum_{j=1}^{i} n_j.c + n_i.w_i$ for $i \geq 1$

These equations are illustrated by Figures 1.4, 1.5 and 1.6. If we let $c_0 = 0$ and $c_i = c$ for $i \geq 1$, we can make this equation more homogeneous: $T_i = \sum_{j=0}^{i} n_j.c_j + n_i.w_i$ for $i \geq 0$.

By definition, the total completion time T is equal to:

$$T = \max_{0 \leq i \leq p} \left(\sum_{j=0}^{i} n_j.c_j + n_i.w_i \right) \tag{1.1}$$

If we rewrite Equation 1.1 as

$$T = n_0.c_0 + \max \left(n_0.w_0, \max_{1 \leq i \leq p} \left(\sum_{j=0}^{i} n_j.c_j + n_i.w_i \right) \right),$$

we recognize an optimal sub-structure for the distribution of $W_{total} - n_0$ among processors P_1 to P_p. This remark allow to easily find a solution for $n_0, \ldots, n_p$ using dynamic programmation. Such a solution is given in Table 1.2.

Nevertheless, this solution is not really satisfying and suffers from several drawbacks. First, we do not have a closed form of the solution (of the n_i's nor of T). Moreover, the order of the processors during the distribution is fixed (the master communicates with P_1, then with P_2, P_3, and so on). Since powers of processors are *a priori* different, this order could be sub-optimal, and this solution does not help us to find the right order among the $p!$ possible orders. There are by far too many possible orders to try an exhaustive search. Furthermore, the time complexity of this solution is $W_{total}^2 \times p$, so the time

Figure 1.4: M computes and sends data to P_1.

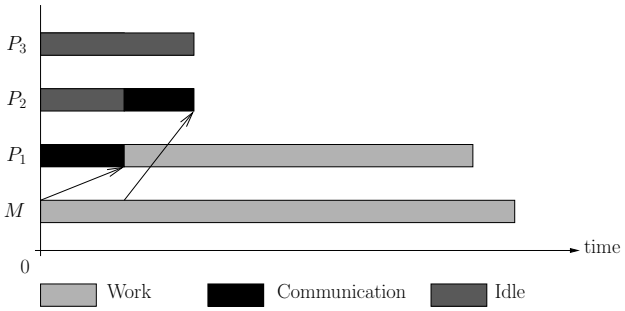

Figure 1.5: M and P_1 compute, M sends data to P_2.

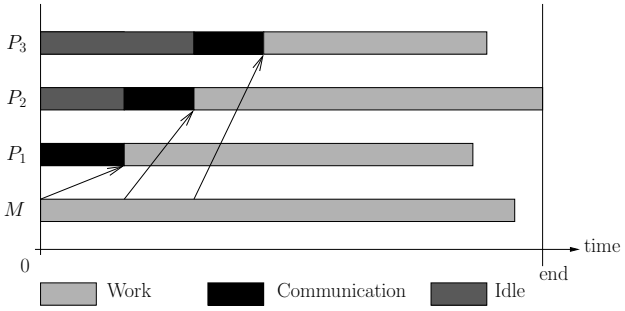

Figure 1.6: Complete schedule.

```
 1: solution[0, p] ← cons(0, NIL); cost[0, p] ← 0
 2: for d ← 1 to W_total do
 3:    solution[d, p] ← cons(d, NIL)
 4:    cost[d, p] ← d · c_p + d · w_p
 5: end for
 6: for i ← p − 1 downto 1 do
 7:    solution[0, i] ← cons(0, solution[0, i + 1])
 8:    cost[0, i] ← 0
 9:    for d ← 1 to W_total do
10:       (sol, min) ← (0, cost[d, i + 1])
11:       for e ← 1 to d do
12:          m ← e · c_i + max(e · w_i, cost[d − e, i + 1])
13:          if m < min then
14:             (sol, min) ← (e, m)
15:          end if
16:       end for
17:       solution[d, i] ← cons(sol, solution[d − sol, i + 1])
18:       cost[d, i] ← min
19:    end for
20: end for
21: return (solution[W_total, 1], cost[W_total, 1])
```

Table 1.2: Solution for the classical approach, using dynamic programming

to decide the right values of the n_is can be greater than the time of effective computation ! Finally, if W_{total} is slightly changed, we cannot reuse the previous solution to obtain a new distribution of the n_is and we have to do the entire computation.

As we can see, even if we an algorithm, which gives us a good solution in polynomial time, we can ask for a better way to solve our problem. Then, let us call the help of the Divisible Load Theory. If we have around $800,000$ tasks for 10 processors, there are roughly $80,000$ tasks on each processor, in the case of identical processors. So, even if we have a rational solution (i.e., a non-integer number of tasks on each processor), we might afford the extra-cost of rounding this solution to obtain integer numbers of tasks and then a valid solution, especially if we can overcome the previous limitations.

1.1.3 Divisible load approach

As said in introduction, the main principle of the Divisible Load Theory is to relax the integer constraint on the number of tasks on each worker. This simple idea can lead to very good results, even if we loose a bit of precision in the solution. Now, let us examine our problem, using this relaxation: instead of a integer number n_i of tasks, processor P_i (with $0 \leq i \leq p$) will compute a fraction α_i of the total load W_{total}, where $\alpha_i \in \mathbb{Q}$. The number of tasks allocated to P_i is then equal to $n_i = \alpha_i W_{total}$ and we have the constraint $\sum_{i=0}^{p} \alpha_i = 1$, ensuring that the whole set of tasks will be computed.

1.1.3.0.1 Bus-shaped network In this paragraph, we keep exactly the same model as before: we still have a bus-shaped network with heterogeneous workers, and data are distributed to workers in a single message, following a linear cost model. Equation 1.1 can be easily translated in Equation 1.2 to use our brand new notation.

$$T = \max_{0 \leq i \leq p} \left(\sum_{j=0}^{i} \alpha_j.c_j + \alpha_i.w_i \right) W_{total} \tag{1.2}$$

Using this equation, we can show several important results in optimal solution:

- all participating processors end their work at the same time in Lemma 1,
- all processors are working in Lemma 2,
- the master processor should be the fastest one but the order of others processors is not important in Lemma 3.

These three lemmas are summarized in Theorem 1.

Lemma 1 *In an optimal solution, all processors end their processing at the same time.*

PROOF Consider any solution, such that at least two workers do not finish their work at the same time. Without any loss of generality, we can assume that these two processors are P_i and P_{i+1} (with $i \geq 0$) and such that either P_i or P_{i+1} finishes its work at time T (the total completion time).

1.1.3.0.2 $i \geq 1$ **and** $T_i < T_{i+1}$ The first case to consider is $T_i < T_{i+1}$, i.e. P_i finishes earlier than P_{i+1}. We decrease the fraction α_{i+1} allocated to P_{i+1} by a size ε, and thus we increase the fraction α_i allocated to P_i by the same fraction ε, as illustrated by Figure 1.9. Since we want P_i and P_{i+1} to finish at the same time, we have the following equations:

$$\begin{aligned}
\left(\sum_{j=0}^{i} \alpha_j.c_j + \alpha_i.w_i + (w_i + c_i)\,\varepsilon\right) W_{total} &= \left(\sum_{j=0}^{i+1} \alpha_j.c_j + \alpha_{i+1}.w_{i+1} - (w_{i+1} + c_{i+1})\,\varepsilon\right) W_{total} \\
\Leftrightarrow (w_i + c_i)\,\varepsilon &= \alpha_{i+1}.c_{i+1} + \alpha_{i+1}.w_{i+1} - (w_{i+1} + c_{i+1})\,\varepsilon \\
\Leftrightarrow \varepsilon\,(w_i + c_i + w_{i+1} + c_{i+1}) &= \alpha_{i+1}.c_{i+1} + \alpha_{i+1}.w_{i+1} \\
\Leftrightarrow \varepsilon &= \tfrac{\alpha_{i+1}.c_{i+1} + \alpha_{i+1}.w_{i+1}}{w_i + c_i + w_{i+1} + c_{i+1}}
\end{aligned}$$

Since we have $c_{i+1} = c_i = c$, the communication time is the same for processors after P_{i+1}. ε is positive, then, in this new solution, both P_{i+1} and P_i finish their work strictly earlier than T_{i+1}

1.1.3.0.3 $i \geq 1$ **and** $T_{i+1} < T_i$ A similar reasoning leads to a strictly better solution, with a decrease of the amount of work on P_{i+1} and an increase of the work of P_i.

1.1.3.0.4 $T_0 < T_1$ This case is very similar to the first one, excepting that the communication time is just smaller for others processors.

1.1.3.0.5 $T_0 > T_1$ Without any loss of generality, we can assume that all others processors finish their work strictly before P_0 (by repeatedly using the first three cases). Let ν be equal to $T_0 - \max_{1 \leq i \leq p} T_i$. Then we decrease the amount of work on P_0 by a fraction ε and we increase the work of P_1 by the same fraction. The completion times T_i of processors $P_2, \ldots, P_p$ are increased by $c.\varepsilon.W_{total}$ and the completion time of P_1 is increased by $(c + w_1)\,.\varepsilon.W_{total}$. If we choose ε such that we have $\max\left(c.\varepsilon.W_{total}, (c + w_1)\,.\varepsilon.W_{total}\right) < \nu$ (i.e., $\varepsilon < \frac{\nu}{(c+w_1).W_{total}}$), then P_0 finish its work strictly earlier than T because it has less work, and others processors still finish earlier than T because the amount of work on P_1 is not too increased.

Then, if in a given solution, at least two processors do not finish their work at the same time, we can in all cases exhibit a strictly better solution.

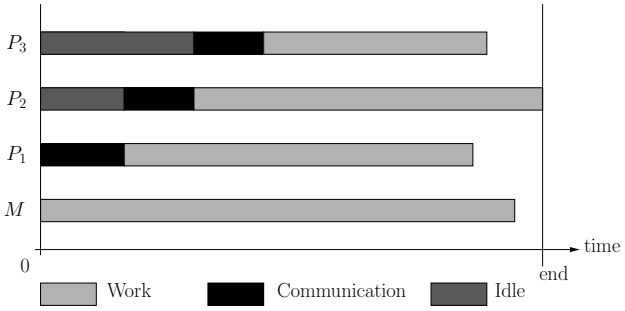

Figure 1.7: P_1 finishes earlier than P_2.

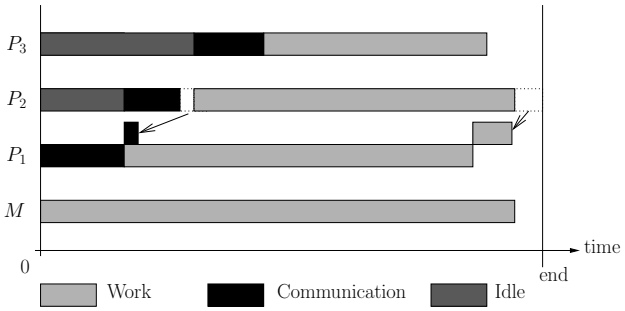

Figure 1.8: A fraction of the load allocated to P_2 is given to P_1.

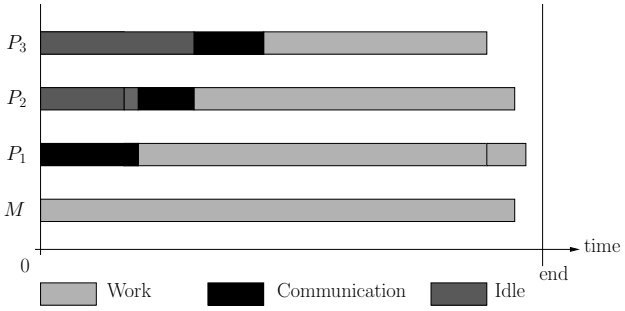

Figure 1.9: This new schedule is strictly better.

We conclude that all processors do have the same end time in any optimal solution. □

Lemma 2 *All processors are enrolled in an optimal solution.*

PROOF This is a direct consequence of Lemma 1. □

Lemma 3 *If we can choose the master processor, it should be the fastest processor, but the order of others processors has no importance.*

PROOF Let us consider any optimal solution $(\alpha_0, \alpha_1, \ldots, \alpha_p)$. By using Lemma 1, we know that $T = T_0 = T_1 = \ldots = T_p$. Thus the following equations:

- $T = \alpha_0.w_0.W_{total}$,
- $T = \alpha_1.(c + w_1).W_{total}$ and then $\alpha_1 = \frac{w_0}{c+w_1}\alpha_0$,
- $T = (\alpha_1.c + \alpha_2.(c + w_2))\,W_{total}$ and then $\alpha_2 = \frac{w_1}{c+w_2}\alpha_1$,
- for all $i \geq 1$, we derive $\alpha_i = \frac{w_{i-1}}{c+w_i}\alpha_{i-1}$.

Thus, we have $\alpha_i = \prod_{j=1}^{i} \frac{w_{j-1}}{c+w_j}\alpha_0$ for all $i \geq 0$. We know that $\sum_{i=0}^{p} \alpha_i = 1$, so we can deduce the complete value of α_0, and thus we have closed formulas for all α_is.

Using these formulas, we are able to answer to both questions. First, we show that we should choose the fastest processor as the master one (when we can decide who is the master processor). Assuming the total completion time is still T and that P_0 is slower than P_1 (i.e., $w_0 > w_1$), we have for processors P_0 and P_1:

- $\alpha_0.w_0.W_{total} = T$ and then $\alpha_0 = \frac{1}{w_0}\frac{T}{W_{total}}$,
- $(\alpha_1.w_1 + \alpha_1.c).W_{total} = T$ and then $\alpha_1 = \frac{1}{c+w_1}\frac{T}{W_{total}}$.

Then the total amount of work done by both processors is equal to:

$$\alpha_0 + \alpha_1 = \frac{c + w_0 + w_1}{w_0.c + w_0.w_1}$$

If we exchange P_0 and P_1 (the faster P_1 becomes the master processor), the total amount of work is then equal to:

$$\alpha_0 + \alpha_1 = \frac{c + w_1 + w_0}{w_1.c + w_1.w_0}$$

Since $w_1 < w_0$, this sum is greater than the previous one: during the same time, we can process a greater amount of work if the fastest processor is the master.

Now, we can apply this reasonning to processors P_i and P_{i+1}, where P_i is not the master processor (i.e., $1 \leq i \leq p-1$). We have the two following equations 1.3 and 1.4:

$$T = T_i = \left(\left(\sum_{j=1}^{i-1} \alpha_j.c \right) + \alpha_i.w_i + \alpha_i.c \right).W_{total} \tag{1.3}$$

$$T = T_{i+1} = \left(\left(\sum_{j=1}^{i-1} \alpha_j.c \right) + \alpha_i.c + \alpha_{i+1}.w_{i+1} + \alpha_{i+1}.c \right).W_{total} \tag{1.4}$$

With $K = \frac{T - W_{total} \cdot \left(\sum_{j=1}^{i-1} \alpha_j.c \right)}{W_{total}}$, we have

$$\alpha_i = \frac{K}{w_i + c}$$

$$\alpha_{i+1} = \frac{K - \alpha_i.c}{w_{i+1} + c}$$

The total fraction of work processed by these two processors is equal to:

$$\alpha_i + \alpha_{i+1} = \frac{K}{w_i + c} + \frac{K}{w_i + c} - \frac{c.K}{(w_i + c)(w_{i+1} + c)}$$

We see that this expression is symmetric in w_i and w_{i+1}, and we can conclude that the communication order has no importance on the quality of a solution, even if it is contrary to the intuition. □

Theorem 1 *For divisible loads applications on bus-shaped networks, we should select the fastest processor as the master; the order of the communications with others processors has no impact on the quality of a solution. All processors participate to the work and they finish simultaneously. Closed-form formulas give the fraction of the load allocated to each processor.*

1.1.3.0.6 Star-shaped network The bus-shaped network model can be seen as a particular case of the more general star-shaped network, with homogeneous communications. Now, we focus our attention on such star-shaped networks: every worker is linked to M through a different communication link as shown in Figure 1.10, processors and communication links have different characteristics.

Notations are the same as the previous ones and are illustrated by Figure **??**: we have a master processor M, and p workers $P_1, \ldots, P_p$. Master sends a unit-size message to P_i (with $1 \leq i \leq p$) in time c_i, and P_i processes it in time w_i. The total amount of load to compute is equal to W_{total}, and P_i receives a fraction α_i of this load (with $\alpha_i \in \mathbb{Q}$ and $\sum_{i=1}^{p} \alpha_i = 1$).

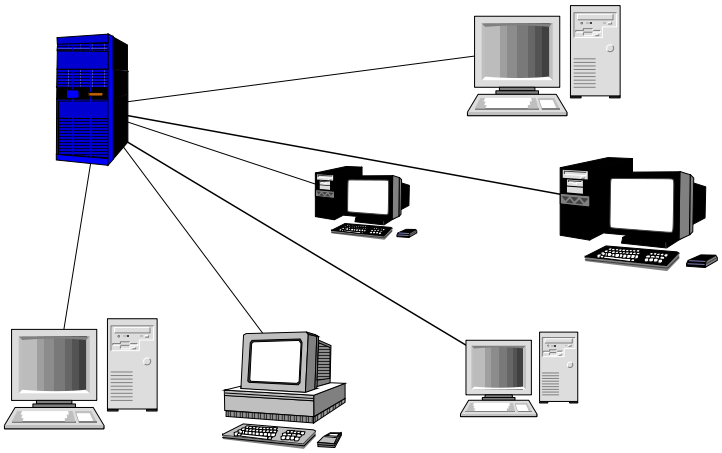

Figure 1.10: Example of star-shaped network.

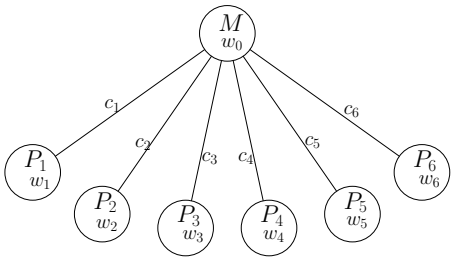

Figure 1.11: Theoritical model of a star-shaped network.

We assume that the master does not help to the computation, but if this assertion is not verified, we can add a virtual processor P_0, with the same speed w_0 as the master and with instantaneous communications: $c_0 = 0$. As before, M sends a single message to each worker, and it can communicate to at most one worker at a time, following the one-port model.

This new model seems to be a simple extension of the previous one, and we will check whether our previous lemmas are still valid. After the presentation of useful equations, we will show that the following lemmas are true for any optimal solution:

- all active workers must be served in the non-decreasing order of c_is in Lemma 4,
- all processors participate to the work in Lemma 5,
- all workers end their work at the same time in Lemma 6,
- an optimal solution can be computed using a linear program in Lemma 7.

As for the bus-shaped network model, all these results are summarized in Theorem 2.

Lemma 4 *In an optimal solution, all active workers must be served in the non-decreasing order of c_is*

PROOF We use the same method as for proving Lemma 3: we consider two workers P_i and P_{i+1} (with $1 \leq i \leq p-1$) and we check whether the total of work which can be done in a given time T by them is dependent of their order.

For these two processors, we have the following Equations 1.5 and 1.6, similar to Equations 1.3 and 1.4 seen in the case of a bus-shaped network:

$$T = T_i = \left(\left(\sum_{j=1}^{i-1} \alpha_j.c_j \right) + \alpha_i.w_i + \alpha_i.c_i \right).W_{total} \tag{1.5}$$

$$T = T_{i+1} = \left(\left(\sum_{j=1}^{i-1} \alpha_j.c_j \right) + \alpha_i.c_i + \alpha_{i+1}.w_{i+1} + \alpha_{i+1}.c_{i+1} \right).W_{total} \tag{1.6}$$

With $K = \frac{T - W_{total} \cdot \left(\sum_{j=1}^{i-1} \alpha_j.c_j \right)}{W_{total}}$, previous Equations 1.5 and 1.6 can be respectively rewritten as:

$$\alpha_i = \frac{K}{w_i + c_i}$$

$$\alpha_{i+1} = \frac{K - \alpha_i.c_i}{w_{i+1} + c_{i+1}}$$

Time needed by communications can be written as:

$$(\alpha_i.c_i + \alpha_{i+1}.c_{i+1}).W_{total} = \left(\frac{c_i}{w_i + c_i} + \frac{c_{i+1}}{w_{i+1} + c_{i+1}}\right).K - \frac{K.c_i.c_{i+1}}{(w_i + c_i)(w_{i+1} + c_{i+1})}$$

We see that the equation is symmetric and the communication time is completely independent of the order of the two communications.

The total fraction of work is equal to:

$$\alpha_i + \alpha_{i+1} = \left(\frac{1}{w_i + c_i} + \frac{1}{w_{i+1} + c_{i+1}}\right).K - \frac{K.c_i}{(w_i + c_i)(w_{i+1} + c_{i+1})}$$

Contrary to the communication time, the fraction of the load done by processors P_i and P_{i+1} depends on the order of the communications. If we serve the fastest-communicating processor first, the fraction of the load processed by P_i and P_{i+1} can be increased without increasing the communication time for others workers. In others terms, in any optimal solution, participating workers should be served by non-decreasing values of c_i. □

Lemma 5 *In any optimal solution, all processors participate to the work.*

PROOF Let us consider an optimal solution $(\alpha_1, \ldots, \alpha_p)$ and assume that at least one processor remains idle during the whole computation. Without any loss of generality, we can also assume that communications are served in the order $(P_1, \ldots, P_p)$. We denote by k the greatest index such that $\alpha_k = 0$ (i.e., P_k is the last processor which is kept idle during the computation). We have two cases to consider:

- $k < p$
  By definition, P_p is not kept idle and thus $\alpha_p \neq 0$. We know that P_p is the last processeor to communicate with the master, and then there is no communication during the last $\alpha_p.w_p.W_{total}$ time units.

  Therefore, we can send at least $\frac{\alpha_p.w_p.W_{total}}{c_k+w_k} > 0$ load units to the processor P_k, and it would finish its computation at the same time as P_p. So, we can exhibit a strictly better solution than the previous optimal one, in which P_k was kept idle.

- $k = p$
  We modify the initial solution to give some work to the last processor, without increasing the total completion time. Let k' be the greatest index such that $\alpha_{k'} \neq 0$. By definition, since $P_{k'}$ is the last served processor, there is no communication with the master during at least $a_{k'}.w_{k'}.W_{total}$ time units. As previously, we can give at least $\frac{\alpha_{k'}.w_{k'}.W_{total}}{c_p+w_p} > 0$ load units to P_p and then exhibits a strictly better solution than the previous optimal one.

Then, in all cases, if at least one processor remains idle, we can build a strictly better solution. This is sufficient to demonstrate that in any optimal solution, all processors participate to the work. □

Lemma 6 *In any optimal solution, all workers end their work at the same time.*

PROOF Since this proof requires more complicated arguments, the reader may want to skip it.

Let us consider any optimal allocation $(\alpha_1, \ldots, \alpha_p)$. Thanks to the Lemma 5, we know that all workers participate to the computation and then all α_i are positive. Consider the following linear program:

$$\begin{array}{l} \text{MAXIMIZE} \sum_{i=1}^{p} \beta_i, \\ \text{SUBJECT TO} \\ \forall i, \beta_i \geq 0, \\ \forall i, \sum_{k=1}^{i} \beta_k c_k + b_i w_i \leq T \end{array}$$

The α_i's obviously satisfy the set of constraints above, and from any set of β_i's satisfying the set of inequalities, we can build a valid schedule that processes exactly $\sum_{i=1}^{p} \beta_i$ units of load. Let $(\beta_1, \ldots, \beta_p)$ be any optimal solution to this linear program. By definition, we have $\sum_{i=1}^{p} \alpha_i = \sum_{i=1}^{p} \beta_i$.

We know that one of the extremal solutions $\mathcal{S}_1$ of the linear program is one of the convex polyhedron $\mathcal{P}$ induced by the inequalities TODO ajouter la ref: this means that in the solution $\mathcal{S}_1$, there are at least p inequalities among the $2p$ equalities. If we use Lemma 5, we know that all the β_i's are positive and then this vertex is the solution of the following (full rank) linear system

$$\forall i \in (1, \ldots, p), \sum_{k=1}^{i} \beta_k c_k + \beta_i w_i = T.$$

Thus, we derive that there exists at least one optimal solution, such that all workers finish simultaneously.

Now, let us consider another optimal solution $\mathcal{S}_2 = (\alpha_1, \ldots, \alpha_p)$, different from $\mathcal{S}_2$. Similarly to $\mathcal{S}_1$, $\mathcal{S}_2$ belongs to the polyhedron $\mathcal{P}$. Now, consider the following function f:

$$f : \begin{cases} \mathbb{R} \to \mathbb{R}^p \\ x \mapsto \mathcal{S}_1 + x(\mathcal{S}_2 - \mathcal{S}_1) \end{cases}$$

By construction, we know that $\sum_{i=1}^{p} \beta_i = \sum_{i=1}^{p} \alpha_i$. Thus, with the notation $f(x) = (\gamma_1(x), \ldots, \gamma_p(x))$:

$$\forall i \in (1, \ldots, p), \gamma_i(x) = \beta_i + x(\alpha_i - \beta_i),$$

and therefore

$$\forall x, \sum_{i=1}^{p} \gamma_i(x) = \sum_{i=1}^{p} \beta_i = \sum_{i=1}^{p} \alpha_i.$$

Therefore, all the points $f(x)$ that belong to $\mathcal{P}$ are extremal solutions of the linear program.

Since $\mathcal{P}$ is a convex polyhedron ad both $\mathcal{S}_1$ and $\mathcal{S}_2$ belong to $\mathcal{P}$, then $\forall 0 \leq x \leq 1, f(x) \in \mathcal{P}$. Let us denote by x_0 the largest value of $x \geq 1$ such that $f(x)$ still belongs to $\mathcal{P}$: at least one constraint of the linear program is an equality in $f(x_0)$, and this constraint is not satisfied for $x > x_0$. We know that this constraint cannot be one of the upper bounds: otherwise, this constraint would be an equality along the whole line $(\mathcal{S}_2, f(x_0))$, and would remain an equality for $x > x_0$. Hence, the constraint of interest is one of the lower bounds. In other terms, there exists an index i, such that $\gamma_i(x_0) = 0$. This is in contradiction with Lemma 5 and with the fact that the γ_i's correspond to an optimal solution of the problem.

We can conclude that $\mathcal{S}_1 = \mathcal{S}_2$, and thus there exists an unique optimal solution, and in this solution, all workers finish simultaneously their work. □

Lemma 7 *We can find a closed-form formula for the fraction of the total load allocated to processor i ($1 \leq i \leq p$):*

$$\beta_i = \frac{\frac{1}{c_i+w_i} \prod_{k=1}^{i-1} \left(1 - \frac{c_k}{c_k+w_k}\right)}{\sum_{j=1}^{p} \left(\frac{1}{c_j+w_j} \prod_{k=1}^{j-1} \left(1 - \frac{c_k}{c_k+w_k}\right)\right)}$$

PROOF Thanks to the previous lemmas, we know that all workers participate to the computation (Lemma 5) and have to be served by non-decreasing values of c_i (Lemma 4) and that all workers finish simultaneously (Lemma 6).

Without any loss of generality, we assume that $c_1 \leq c_2 \leq \ldots \leq c_p$. By definition, the completion time of the i-th processor is given by:

$$\forall i \in (1, \ldots, p), T_i = T = W_{total} \left(\sum_{k=1}^{i} \beta_k c_k + \beta_i c_i \right). \tag{1.7}$$

For the first processor, Equation 1.7 can be written as:

$$T = W_{total} \beta_1 (c_1 + w_1) \Leftrightarrow \beta_1 = \frac{T}{W_{total}} \frac{1}{c_1 + w_1}$$

Assume that for $i \geq 1$, we have the following equation

$$\beta_i = \frac{T}{W_{total}} \frac{1}{c_i + w_i} \prod_{k=1}^{i-1} \left(1 - \frac{c_k}{c_k + w_k}\right). \tag{1.8}$$

Thanks to Equation 1.7, we have

$$\beta_i = \frac{T}{W_{total}} \frac{1}{w_i + c_i} \left(1 - \sum_{k=1}^{i-1} \beta_k c_k\right)$$

and

$$\begin{aligned}
T_{i+1} = \quad T &= W_{total} \left(\textstyle\sum_{k=1}^{i} \beta_k c_k + \beta_{i+1} (w_{i+1} + c_{i+1})\right) \\
\Leftrightarrow \beta_{i+1} &= \tfrac{T}{W_{total}} \tfrac{1}{w_{i+1}+c_{i+1}} \left(1 - \textstyle\sum_{k=1}^{i} \beta_k c_k\right) \\
\Leftrightarrow \beta_{i+1} &= \tfrac{T}{W_{total}} \tfrac{1}{w_{i+1}+c_{i+1}} \left(\left(1 - \textstyle\sum_{k=1}^{i-1} \beta_k c_k\right) - \beta_i c_i\right) \\
\Leftrightarrow \beta_{i+1} &= \tfrac{c_i+w_i}{c_{i+1}+w_{i+1}} \left(\left(\tfrac{T}{W_{total}} \tfrac{1}{w_i+c_i} \left(1 - \textstyle\sum_{k=1}^{i-1} \beta_k c_k\right)\right) - \beta_i \tfrac{c_i}{c_i+w_i}\right) \\
\Leftrightarrow \beta_{i+1} &= \tfrac{c_i+w_i}{c_{i+1}+w_{i+1}} \beta_i \left(1 - \tfrac{c_i}{c_i+w_i}\right)
\end{aligned}$$

Time to use Equation 1.8:

$$\beta_{i+1} = \frac{T}{W_{total}} \frac{1}{c_{i+1} + w_{i+1}} \prod_{k=1}^{i} \left(1 - \frac{c_k}{c_k + w_k}\right).$$

We still do not know the total completion time T, but we can use the linear model for communication and computation costs. We define for any processor i $(1 \leq i \leq p)$ $\gamma_i = \frac{1}{c_{i+1}+w_{i+1}} \prod_{k=1}^{i} \left(1 - \frac{c_k}{c_k+w_k}\right)$ and then we have our allocation by normalizing these γ's:

$$\forall 1 \leq i \leq p, \beta_i = \frac{\gamma_i}{\sum_{k=1}^{p} \gamma_k}.$$

Thus, our complete closed-form formula for the fraction of the total load allocated to the i-th worker is:

$$\beta_i = \frac{\frac{1}{c_i+w_i} \prod_{k=1}^{i-1} \left(1 - \frac{c_k}{c_k+w_k}\right)}{\sum_{j=1}^{p} \left(\frac{1}{c_j+w_j} \prod_{k=1}^{j-1} \left(1 - \frac{c_k}{c_k+w_k}\right)\right)}$$

□

Theorem 2 *For divisible loads applications on star-shaped networks, we should order workers by non-decreasing values of c_i. All processors participate to the work and they finish simultaneously. Closed-form formulas give the fraction of the load allocated to each processor.*

We will conclude this section with two remarks.

- If the order of the communications cannot be freely chosen, Lemma 5 is not always true.

- In this section, we considered that the master processor did not participate to the processing. If this condition is not satisfied, we can easily add a virtual worker, with the same processing speed as the master, and a communication time equal to 0. If we can also choose the master processor, the simplest way to find the best choice is to compute the total completion time in all cases: we have only p different cases to check, so this trivial solution remains feasible.

1.2 Several extensions to the divisible load model

In the first part of this chapter, we have shown that th use of the divisible load theory can greatly help to solve scheduling problems, that are hard to solve if we only use classical models.

When we do not use the divisible load theory, we have to use a very simple model (homogeneous communication links and linear cost functions for computations and communications). The divisble load theory changes this already hard problem in a very simple one, and we already saw that this heory allows the problem to remain tractable even with heterogeneous communications.

In one hand, a very realistic model should surely be untractable, and a very simple model has no chance to be realistic. Then the natural question is to find hwo far can we go in extending our model, the problem remaining tractable with the help of the divisible load theory. We show in the next sections several extensions to the basic situation. In Section 1.2.1, we cancel the linear cost model for communications by introducing latencies. We decide to distribute chunks to processors in several messages in Section 1.2.2. Another possible modification is the cancellation of the overlap of communications by computations, as done in Section **??**. In Section **??**, we generalize the star-shaped network by using tree-shaped networks. We still assume that data were only sent from the master to the workers, and that the result of the computation was a negligible communication, and in Section 1.2.3 we removes this assumption. We finish this section by summarize all these results in Section **??**.

1.2.1 Introducing latencies

In the previous section, we use a simple but efficient linear cost model for communications. So, the time needed to transmit a data is perfectly proportionnal to the size of these data. In the real world, we always have a small latency as well a communication header and then a better communication model should be affine. We have to introduce new notations for these latencies: C_i denotes the communication latency paid by worker P_i for any

communication from the master, and W_i denotes the latency corresponding to an initialization of any computation. If P_i has to process a fraction α_i of the total load, then its communication time is equal to $C_i + c_i\alpha_i W_{total}$ and its computation time is equal to $W_i + w_i\alpha_i W_{total}$.

This variant of the problem is much more difficult than the previous one. In fact, its complexity remains an open problem. However, some important results can be shown for any optimal solution:

- all workers end their work at the same time in Lemma 8,
- if the load is large enough, all workers participate to the work and must be served in the non-decreasing order of c_i in Lemma 9,
- an optimal solution can be found using a mixed linear program in Lemma 10.

Lemma 8 *In any optimal solution, all workers have the same completion time.*

PROOF This lemma can be shown using a proof similar to the proof of Lemma 6. A detailed proof is given in TODO(ref à un techreport). □

Lemma 9 *In any optimal solution, and if the load is large enough, all workers participate to the work and must be served in the non-decreasing order of communication time c_i.*

PROOF We want to determine the total amount of work which can be done in a time T, and let us consider any valid solution to this problem. The set of the k active workers is denoted $\mathcal{S} = \{P_{\sigma(1)}, \ldots, P_{\sigma(k)}\}$, where σ is a one-to-one mapping from $[1 \ldots k]$ to $[1 \ldots p]$ and represents the order of communications. The number of processed units of load reachable using this set of workers in this order is denoted by n^{TASK}.

- Consider the following instance of our problem, with k workers $P'_{\sigma(1)}, \ldots, P'_{\sigma(k)}$, such that $\forall i \in \{1, \ldots, k\}, C'_i = 0, W'_i = 0, c'_i = c_i, w'_i = w_i$ (in fact, we are just ignoring all latencies). We will determine the total number of work units n'^{TASK} which can be done in the same time T, and of course this number is greater than the number n^{TASK} obtained with the original platform:
$$n^{TASK} \leq n'^{TASK}.$$
Using Theorem 2, we know that n'^{TASK} is given by a formula of the following form:
$$n'^{TASK} = f(\mathcal{S}, \sigma) \cdot T.$$
The main point is that n'^{TASK} is proportionnal to T.

- Consider now the same platform without latencies, but this time we will determine the number of works units n'^{TASK}, which could be done in a time $T' = T - \sum_{i \in \mathcal{S}}(C_i + W_i)$. n'^{TASK} is clearly smaller than n^{TASK} since it consists in adding all latencies before the beginning of the work:

$$n'^{TASK} \leq n^{TASK}.$$

The previous equality still stands:

$$n'^{TASK} = f(\mathcal{S}, \sigma) \left(T - \sum_{i \in \mathcal{S}} (C_i + W_i) \right).$$

We have $n'^{TASK} \leq n^{TASK} \leq n'^{TASK}$ and then

$$f(\mathcal{S}, \sigma) \left(1 - \frac{\sum_{i \in \mathcal{S}} (C_i + W_i)}{T} \right) \leq \frac{n^{TASK}}{T} \leq f(\mathcal{S}, \sigma).$$

Therefore, when T becomes arbitrarily large, then the throughput of the platform becomes close to the theoritical model without any latency. Thus, when T is sufficiently large, in any optimal solution, all workers participate to the work, and chunks should be sent on the ordering of non-decreasing communication times c_i.

Without any loss of generality, we can assume that $c_1 \leq \ldots \leq c_p$ and then the following linear system returns an asymptotically optimal solution:

$$\forall i \in \{1, \ldots, p\}, \sum_{k=1}^{i} (C_k + c_k \alpha_k W_{total}) + W_i + w_i \alpha_i W_{total} = T.$$

Moreover, when T is sufficiently large, this solution is optimal when all c_i are different, but determine the best way to break ties among workers having the same communication speed remains an open question.

□

Lemma 10 *An optimal solution can be found using a mixed linear program (with a potentially exponential computation cost).*

PROOF In TODOref au papier de Casanova etal, the authors added the resource selection issue to the original linear program given by Drozdowsky in TODO trouver la bonne ref. To address this issue, they added two notations: y_j, which is a binary variable equal to 1 if P_j participates to the work, and $x_{i,j}$, which is a binary variable equal to 1 if P_j is chosen for the i-th communication

from the master.

$MAXIMIZE\ T_f$
$SUBJECT\ TO$
$\forall i \in \{1,\dots,p\}, \alpha_i \geq 0$
$\sum_{i=1}^{p} \alpha_i = 1$
$\forall j \in \{1,\dots,p\}, y_j \in \{0,1\}$
$\forall i,j \in \{1,\dots,p\}, x_{i,j} \in \{0,1\}$
$\forall j \in \{1,\dots,p\}, \sum_{i=1}^{p} x_{i,j} = y_j$
$\forall i \in \{1,\dots,p\}, \sum_{j=1}^{p} x_{i,j} \geq 1$
$\forall j \in \{1,\dots,p\}, \alpha_j \leq y_j$
$\forall i \in \{1,\dots,p\}, \sum_{k=1}^{i-1}\sum_{j=1}^{p} x_{k,j}(C_j + \alpha_j c_j W_{total}) + \sum_{j=1}^{p} x_{i,j}(C_j + \alpha_j c_j W_{total} + W_j + \alpha_j w_j W_{total}) \leq T_f$

Equation 5 implies that P_j is involved in exactly y_j communication. Equation 6 states that that at most one worker is used for the i-th communication. Equation 7 ensures that non-participating workers have no work to process. Equation 8 implies that the worker selected for the i-th communication must wait for the previous communications before starting its own communication and then its computation.

This linear program always has a solution and gives us the selected workers and their fraction of the total load to process in an optimal solution. □

1.2.2 Multiround strategies

Until now, we only used models with a strict one-port communication scheme, and data were transmitted to workers in a single message. Therefore, any worker has to wait while previous workers are communicating with the master before beginning to process its part of the work, and this waiting time can lead to a poor utilization of the platform. A natural solution to quickly distribute some work to every processor is to distribute data in multiple rounds: while the first processor is computing the beginning of its task, we can distribute data to others workers and then we resume the distribution to the first processor. By this way, we hope to overlap communications by computation and thus to increase the throughput of our platform, as represented by Figure 1.12.

If this idea seems to be promising, we have two new questions to answer:

1. how many rounds should we use to distribute the whole load?
2. which size should we allocate to each round?

If we follow the example of the problem using a single-round distribution, we should try to solve this problem without latencies for communications before adding them. In fact, this case is not interesting since it leads to an optimal solution with an infinite number of rounds of size zero. On the contrary, when latencies are added to the model, they prevent the solution to use such an infinite number of rounds, but we saw that problems were by far harder with latencies.

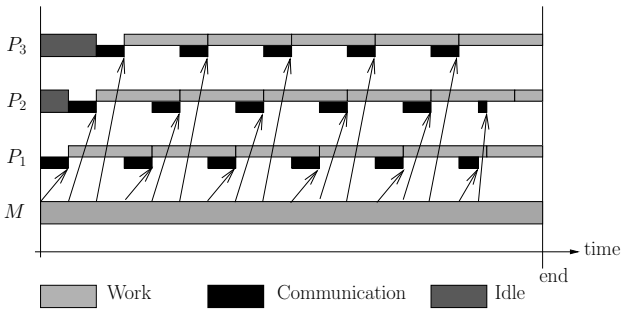

Figure 1.12: Multi-round execution over a bus-shaped platform.

1.2.2.1 Bus-shaped network and homogeneous processors, fixed number of rounds

The simplest case to explore is a bus-shaped network of homogeneous processors, i.e., homogeneous communication links, and when the number of rounds is given to the algorithm.

Intuitively, rounds have to be small to allow a fast start of computations, and have to be large to amortize the cost of latencies. These two contradictory objectives can be merged by using small rounds at the beginning, and then increasing them progressively to amortize paid latencies.

The first works on this topic were done by Bharadwaj et al. using a linear cost model for both communications and computations TODO reference à trouver, and they were followed by Yang and Casanova TODO reference à ajouter using affine models instead linear ones.

Since we only consider homogeneous platforms, we have for any worker i (with $1 \leq i \leq p$) $w_i = w$, $W_i = W$, $c_i = c$ and $C_i = C$. Moreover, R denotes the computation-communication ratio ($R = w/c$) and γ_j denotes the time to compute the chunk j excluding the computation latency: $\gamma_j = \alpha_j \cdot w \cdot W_{total}$. We assume that we distribute the whole load in M rounds of p chunks. For technical reasons, chunks are numbered in the reverse order, from $Mp-1$ (the first one) to 0 (the last one).

Using these notations, we can write the recursion on the γ_j series:

$$\forall j \geq p, W + \gamma_j = \frac{1}{R}(\gamma_{j-1} + \gamma_{j-2} + \ldots + \gamma_{j-p}) + p \cdot C \tag{1.9}$$

$$\forall 0 \leq j < p, W + \gamma_j = \frac{1}{R}(\gamma_{j-1} + \gamma_{j-2} + \ldots + \gamma_{j-p}) + j \cdot C + \gamma_0 \tag{1.10}$$

$$\forall j < 0, \gamma_j = 0 \tag{1.11}$$

Equation 1.9 expresses that a worker j must receive enough data to compute during exactly the time needed for the next p chunks to be communicated,

ensuring no idle time on the communication bus. This equation is of course not true for the last p chunks. Equation 1.10 states that all workers have to finish their work at the same time, following Lemma 8. Finally, Equation 1.11 ensures the correctness of the two previous equations by setting out-of-range terms to 0.

This recursion corresponds to an infinite α_j series, of which the first pM values give the solution to our problem. Using generating functions, we can solve this recursion. Let us call $\mathcal{G}(x)$ the generating function for the series γ_j, i.e., $\mathcal{G}(x) = \sum_{j=0}^{\infty} \gamma_j x_j$. Using Equations 1.9 and 1.10, Yang and Casanova found the value of $\mathcal{G}(x)$ in TODOtoujours la même ref:

$$\mathcal{G}(x) = \frac{(\gamma_0 - p \cdot C)(1 - x^p) + (p \cdot C - W) + C \cdot (\frac{x(1-x^{p-1})}{1-x} - (p-1)x^p)}{(1-x) - x(1-x^p)/R}.$$

The rational expansion methode (TODO ref needed) gives the roots of the polynomial denominator and then the correct values of the γ_j's, and finally, the values of the α_j's. The value of the first term γ_0 is given by the equation $\sum_{j=0}^{Mp-1} \gamma_j = W_{total} \cdot w$.

TODO écrire la même chose sous la forme d'un joli lemme ?

1.2.2.2 Bus-shaped network, computed number of rounds

In the previous section, we assumed that the number of rounds was given to the algorithm, thus we avoided one of the two issues of multiround algorithms. Now, we suppose that the number of chunks has to be computed by the scheduling algorithm as well their respective sizes. As we said before, we have to find a good compromise between a small number of chunks, to reduce the overall cost of latencies, and a large one, to ensure a good overlap of communications by computations.

In fact, finding the optimal number of rounds for such algorithms and affine cost models is still an open question. Nonetheless, Casanova and Yang proposed the Uniform Multi-Round (UMR) algorithm (TODO encore une ref à mettre....). This algorithm is valid in the homogeneous case as well in the heterogeneous case, but we will only look at the homogeneous one for simplicity reasons. To simplify the problem, UMR assume that all chunks sent to workers during the same round have the same size. This constraint can limit the overlap of communications by computations, but it allows to find an optimal number of rounds.

1.2.2.2.1 Optimal number of rounds In this section, α_j denotes the fraction of the load sent to any worker during the j-th round, so there are p chunks of size α_j. The constraint of uniform sizes for chunks of the same round is not used for the last round, allowing the workers to finish simultaneously. To ensure a good utilization of the communication link, authors force the last worker P_p to start its computation immediately after the reception of its data.

This condition can be written as:

$$W + \alpha_j \cdot w \; cdot W_{total} = p \cdot (C + \alpha_{j+1} \cdot c \cdot W_{total}),$$

which leads to:

$$\alpha_j = \left(\frac{c}{p \cdot w}\right)^j (\alpha_0 - \gamma) + \gamma, \tag{1.12}$$

where $\gamma = \frac{1}{w - p \cdot c} \cdot (p \cdot C - W)$. The case $w - p \cdot c$ is simpler and detailed in the original paper TODO ref à ce papier, justement.

With this simple formula, we can give the makespan $\mathcal{M}$ of the complete schedule, which is the sum of the time needed by the worker p to process its data, the total latency of computations and the time needed to send all the chunks during the first round (the $\frac{1}{2}$ factor comes from the non-uniform sizes of the last round, since all workers finish simultaneously):

$$\mathcal{M}(M, \alpha_0) = \frac{W_{total}}{p} + M \cdot W + \frac{1}{2} \cdot p \cdot (C + c \cdot \alpha_0). \tag{1.13}$$

The complete schedule needs to process the entire load, which can be written as:

$$\mathcal{G}(M, \alpha_0) = \sum_{j=0}^{M-1} p \cdot \alpha_j = 0. \tag{1.14}$$

Using these equations, our problem can be expressed as minimizing $\mathcal{M}(M, \alpha_0)$ subject to $\mathcal{G}(M, \alpha_0)$. The Lagrange Multiplier method (TODO ref à trouver) leads to a single equation, which can not be solved analytically but numerically. Several simulations showed that the uniform chunks can reduce performances compared to the first multi-round algorithm when latencies are small but leads to better results when latencies are important. Moreover, the UMR algorithm can be used on heterogeneous platforms, contrary to the previous multi-round algorithm.

1.2.2.2.2 Asymptotically optimal algorithm Clearly, to find an optimal algorithm distributing data to workers in several rounds is hard and is still an open question. Nonetheless, it is possible to design asymptotically optimal algorithms. An algorithm is asymptotically optimal if the ratio of the makespan obtained with a load W_{total} over the optimal makespan with this same load tends to 1 as W_{total} tends to infinity. This approach is coherent with the fact that the Divisible Load Theory already is an approximation well fitted to large workloads.

Theorem 3 *Let consider a star-shaped platform with arbitrary values of computation and communication speeds and latencies allowing the overlap of communications by computations, then there exists a periodic multi-round algorithm, which asymptotically optimal.*

PROOF

The main idea is to look for a periodic schedule: the makespan T is divided into k periods of duration T_p. The initialization and the end of the schedule are sacrified, but the large duration of the whole schedule amortizes this sacrifice. We still have to find a good compromize between small and large periods, and choosing a period length proportionnal to the square-root of the optimal makespan T^* is a good trade-off. The other problem is to choose the participating workers, and was solved by Yand and Casanova using linear programming. If $\mathcal{I} \subseteq \{1, \ldots, p\}$ denotes selected workers, we can write that communication and computation resources are not exceeded during a period of duration T_p:

$$\sum_{i \in \mathcal{I}} (C_i + \alpha_i \cdot c_i W_{total}) \leq T_p, \tag{1.15}$$

$$\forall i \in \mathcal{I}, W_i + \alpha_i \cdot w_i \cdot W_{total} \leq T_p. \tag{1.16}$$

We aim to maximise the average throughput $\rho = \sum_{i \in \mathcal{I}} \frac{\alpha_i \cdot W_{total}}{T_p}$, where $\frac{\alpha_i \cdot W_{total}}{T_p}$ is the average number of load units processed by P_i in one time unit, under the following linear constraints:

$$\begin{cases} \forall i \in \mathcal{I}, & \frac{\alpha_i \cdot W_{total}}{T_p} w_i \leq 1 - \frac{W_i}{T_p} & \textit{(overlap)}, \\ & \sum_{i \in \mathcal{I}} \frac{\alpha_i \cdot W_{total}}{T_p} c_i \leq 1 - \frac{\sum_{i \in \mathcal{I}} C_i}{T_p} & \textit{(1-port model)} \end{cases}$$

This set of constraint can be replaced by the following one, stronger but easier to solve:

$$\begin{cases} \forall i \in \{1, \ldots, p\}, & \frac{\alpha_i \cdot W_{total}}{T_p} w_i \leq 1 - \frac{\sum_{i=1}^{p} C_i + W_i}{T_p} & \textit{(overlap)}, \\ & \sum_{i=1}^{p} \frac{\alpha_i \cdot W_{total}}{T_p} c_i \leq 1 - \frac{\sum_{i=1}^{p} C_i + W_i}{T_p} & \textit{(1-port model)} \end{cases} \tag{1.17}$$

Without any loss of generality, assume that $c_1 \leq c_2 \leq \ldots \leq c_p$ and let q be the largest index, such that $\sum_{i=1}^{q} \frac{c_i}{w_i} \leq 1$. Let ε be equal to $1 - \sum_{i=1}^{q} \frac{c_i}{w_i}$ if $q < p$, to 0 otherwise. Then the optimal throughput is realized with

$$\begin{aligned} \forall 1 \leq i \leq q, \quad \frac{\alpha_i \cdot W_{total}}{T_p} &= \frac{1}{c_i} \left(1 - \frac{\sum_{i=1}^{p} C_i + W_i}{T_p}\right) \\ \frac{\alpha_{q+1} \cdot W_{total}}{T_p} &= \left(1 - \frac{1}{T_p} \sum_{i=1}^{p} (C_i + W_i)\right) \left(\frac{\varepsilon}{c_{q+1}}\right) \\ \forall q+2 \leq i \leq p \qquad \alpha_i &= 0 \end{aligned}$$

and it is equal to

$$\rho = \sum_{i=1}^{p} \frac{\alpha_i \cdot W_{total}}{T_p} = \left(1 - \frac{\sum_{i=1}^{p} C_i + W_i}{T_p}\right) \rho_{\text{opt}} \textit{with} \rho_{\text{opt}} = \sum_{i=1}^{q} \frac{1}{w_i} + \frac{\varepsilon}{c_{q+1}} \tag{1.18}$$

To prove the asymptotic optimality of this algorithm, we need an upper bound of the optimal throughput ρ^*, which can be obtained by removing all latencies (i.e., $G_i = 0$ and $W_i = 0$ for any worker i). Equation 1.18 states

that we have $\rho^* \leq \rho_{\text{opt}}$. If we call T^* the optimal time needed to process B load units, then we have

$$T^* \geq \frac{B}{\rho^*} \geq \frac{B}{\rho_{opt}}.$$

Let denote T the time needed by the proposed algorithm to compute the same workload B. The first period is dedicated to communications and is lost for processing, so $k = \left\lceil \frac{B}{\rho \cdot T_p} \right\rceil + 1$ periods are required for the whole computation.

We have $T = k \cdot T_p$, therefore,

$$T \leq \frac{B}{\rho} + 2 \cdot T_p \leq \frac{B}{\rho_{opt}} \left(\frac{1}{1 - \sum_{i=1}^{p} \frac{C_i + W_i}{T_p}} \right) + 2 \cdot T_p,$$

and therefore, if $T_p \geq 2 \cdot \sum_{i=1}^{p} C_i + W_i$,

$$T \leq \frac{B}{\rho_{\text{opt}}} + 2 \cdot \frac{B}{\rho_{\text{opt}}} \sum_{i=1}^{p} \frac{C_i + W_i}{T_p} + 2 \cdot T_p$$

and if T_p is equal to $\sqrt{\frac{B}{\rho_{\text{opt}}}}$,

$$\frac{T}{T^*} \leq 1 + 2 \left(\sum_{i=1}^{p} (C_i + W_i) + 1 \right) \frac{1}{\sqrt{T^*}} = 1 + O\left(\frac{1}{\sqrt{T^*}} \right)$$

That suffices to show the asymptotic optimality of the proposed algorithm. □

1.2.2.2.3 Maximum benefits of multi-round algorithms

1.2.3 Return Messsages

In all previous sections, we assumed that computations required a lot of data as input, but that they produced a negligible ouput, so we do not take care of its transmission. This assumption could be false since computations could produce large outputs like cryptographic keys. On the contrary, we suppose in this section that we have return messages, in order to see how the previous results are changed, or not.

In the general case, there is no correlation between input and output sizes, but we simplify the problem by assuming the same size for input and output messages. In others words, if M needs $c_i \alpha_i W_{total}$ time units to send the input to worker P_i, P_i needs the same time to send the result back to M after having completed its computation. The communication medium is supposed to be bi-directionnal (as most of network cards are now full-duplex), so the master M can simultaneously send and receive data.

In our first model with linear cost models and distribution in a single round, all workers participated to the work and we were able to find an optimal order

to distribute data. If we allow return messages, we have two new issues : the order of return messages could be different from the distribution order and several workers could remain idle during the whole computation. Two simple ordering strategies are the FIFO strategy (return messages are sent in the same order as the input messages) and the LIFO strategy (return messages are sent in reverse order). In fact, several examples can (TODO faire des jolis graphiques avec de tel exemples) be exhibited, such that the optimal order for return messages is neitheir FIFO or LIFO.

Moreover, the distribution of data in a single round induces long waiting times, and a multi-round distribution could really improve this drawback. Regrettably, any optimal multi-round distribution for the linear cost model uses an infinite number of rounds. Thus, affine cost models is required to have realistic solutions, but the problem becomes very hard to solve or even to approximate.

1.3 Final words

In this chapter, we paid attention to the Divisible Load Theory, a simple and common relaxation. An common example of application is used as a guideline. Without relaxation, this example already is a tractable problem. However, the known solution to this problem suffers from several drawbacks. Moreover the linear model used for the communication and computation costs and the homogeneous communication model limit the representativeness. We show how we can use the DLT to simplify the problem and to solve it. After having simplified the problem with this relaxation, we can use a more featured model by adding heterogeneous communication links or latencies in communications and computations.