The Master-Slave Paradigm with Heterogeneous Processors

Olivier Beaumont, Arnaud Legrand, and Yves Robert, *Member*, *IEEE*

Abstract—In this paper, we revisit the master-slave tasking paradigm in the context of heterogeneous processors. We assume that communications are handled by a bus and, therefore, at most one communication can take place at a given time step. We present a polynomial algorithm that gives the optimal solution when a single communication is needed before the execution of the tasks on the slave processors. When communications are required both before and after the processing of the tasks, we show that the problem is strongly NP-Complete. In this case, we present a guaranteed approximation algorithm. Finally, we present asymptotically optimal algorithms when communications are required before the processing of each task, or both before and after the processing of each task.

Index Terms—Heterogeneous processors, master-slave tasking, communication, matching, complexity.

1 Introduction

Master-slave tasking is a simple yet widely used technique to execute independent tasks under the centralized supervision of a control processor. In the standard implementation of master-slave, the tasks are executed by identical processors (the slaves). We revisit the master-slave paradigm in the framework of heterogeneous computing resources: Slave processors have different computation speeds. We present several scenarios to model the communication pattern between the master and the slaves. In all cases, such communications will take place in exclusive mode on a dedicated hardware resource (such as a serial bus), i.e., *at most one communication will take place between the master and one slave at any time step*.

To give a single motivation, this framework applies to any Monte Carlo simulation where large numbers of identical and independent simulations are run for different values of the random number generator seed. Monte Carlo simulations are widely used in various areas such as cellular microphysiology [20], reactor simulations [21], or studies on conformations of proteins [19].

The rest of the paper is organized as follows: In Section 2, we state four different variants of the master-slave problem: 1) with communications only before dispatching the tasks, 2) with communications both before and after processing the tasks, 3) with communication before each task processing, and 4) with communication both before and after each task processing. We give, in Section 3, a polynomial time algorithm that solves the first problem. The second problem is intrinsically more difficult and we prove in Section 4 that it is strongly NP-Complete; we also prove a guaranteed approximation algorithm for the second problem in Section 4, and we report some simulation results. We present asymptotically optimal algorithms when communications are required before each task (third problem) in Section 5, and when communications are required both before and after each task (fourth problem) in Section 6. We briefly survey related work in Section 7. Finally, we give some remarks and conclusions in Section 8.

- *A. Legrand and Y. Robert are with LIP, UMR CNRS-ENS Lyon-INRIA 5668, Ecole Normale Superieure de Lyon, 69364 Lyon Cedex 07, France. E-mail: {Arnaud.Legrand, Yves.Robert}@ens-lyon.fr.*
- *O. Beaumont is with LaBRI, UMR CNRS 5800, Domaine Universitaire, 33405 Talence Cedex, France. E-mail: Olivier.Beaumont@labri.fr.*

Manuscript received 1 Apr. 2001; revised 22 July 2002; accepted 31 Mar. 2003.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 113908.

2 Problem Statement

The target architectural platform is represented in Fig. 1. The master M and the p slaves $P_1, P_2, \ldots, P_p$ communicate through a shared medium, typically a bus, that can be accessed only in exclusive mode. In this context, at a given time-step, at most one slave processor can communicate with the master, either to receive data or to send results back.

We assume that there is a pool of independent tasks to be processed by the p slaves. All tasks are of same-size, that is, they represent the same amount of processing. Tasks are considered to be atomic (execution cannot be preempted once initiated). Processors are heterogeneous; more precisely, slave P_i requires t_i units of time to process a single task. We say that t_i is the "cycle-time" of processor P_i. Each P_i will execute c_i tasks (where c_i is to be determined) from the pool. Regardless of the hypotheses concerning communication costs, there are two (related) optimization problems:

MinTime(C). Given a total number of tasks C, determine the best allocation of tasks to slaves, i.e., the allocation $\mathcal{C} = \{c_1, c_2, \ldots, c_p\}$ s.t. $\sum_{i=1}^{p} c_i = C$ and which minimizes the total execution time.

MaxTasks(T). Given a time bound T, determine the best allocation of tasks to slaves, i.e., the allocation $\mathcal{C} = \{c_1, c_2, \ldots, c_p\}$ s.t. all processors complete their execution within T units of time and $\sum_{i=1}^{p} c_i = C$ is maximized.

In the paper, we concentrate on solving the second problem MaxTasks(T). Given the solution to this problem, we find a solution to MinTime(C) by using binary search on T, calling MaxTasks(T) several times, and returning the

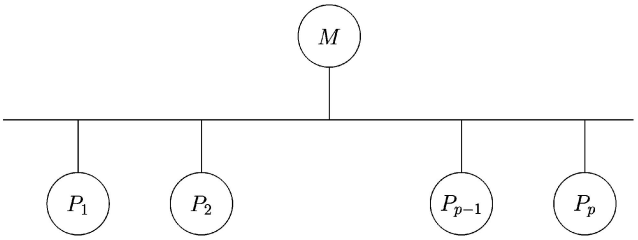

Fig. 1. The target master-slave architecture.

smallest value of T for which the answer is at least C. More precisely, to solve MinTime(C), we first determine an upper bound T, for instance, by assigning all C tasks to the fastest processor, and by rounding up the execution time obtained in this way to the closest power of 2. Next, we call MaxTasks($T/2$): If the answer is greater than or equal to C, we call MaxTasks with the value $T/4$; otherwise, we call MaxTasks with the value $3T/4$, and we proceed recursively.

We now state some specific hypotheses for the communication costs. For each modeling of these communication costs, we analytically formulate the MaxTasks(T) problem.

2.1 Without Any Communication Cost

Assume first that there is no communication cost at all. The solution of problem MaxTasks(T) is straightforward. Because each slave processor can work independently during T time-steps, the only constraint for P_i writes $c_i t_i \leq T$, or $c_i \leq \frac{T}{t_i}$. Because the c_i must be integers, letting $c_i = \left\lfloor \frac{T}{t_i} \right\rfloor$ for all i, $1 \leq i \leq p$, clearly defines a feasible and optimal solution.

In this simple case, note that problem MinTime(C) can also be solved directly, using the greedy algorithm described in [3] to distribute independent tasks to heterogeneous processors.

2.2 With an Initial Scattering of Data

The formulation of this problem is taken from Andonie et al. [1], who study the implementation of distributed back-propagation neural networks on heterogeneous networks of workstations, using the PVM library [11]. The training of the neural network is divided into computational phases. At each phase, the training pattern is distributed among the slaves, which are different-speed processors. Before executing any task, each slave must receive some data file from the master processor. Because the communication medium is exclusive, it is not relevant to distinguish whether the data file is the same for all slaves (then, the master executes a broadcast operation), or whether it is different (then, the master executes a scatter operation): We only assume that each slave must receive the same amount of data, and that each reception costs t_{com} units of time. In the model of Andonie et al. [1], there is no communication cost paid to send the results back to the master. In general, when the slaves compute "yes/no" results, the cost of returning the results may well be neglected in front of the cost of the initial scatter and/or of the computations. Note that we deal with another model, including communication costs both before and after the tasks, in Section 2.3.

Due to the constraint on the communication medium, the p messages will be sent one after the other. Obviously, it cannot hurt to send the messages as soon as possible, i.e., at time steps 0, t_{com}, $2t_{\text{com}}$, $\ldots$, $(p-1)t_{\text{com}}$. The problem is then to determine the ordering of the p messages, i.e., the permutation σ of $\{1, 2, \ldots, p\}$ such that slave P_i receives the message at time $\sigma(i)t_{\text{com}}$. We are ready to state the optimization problem analytically:

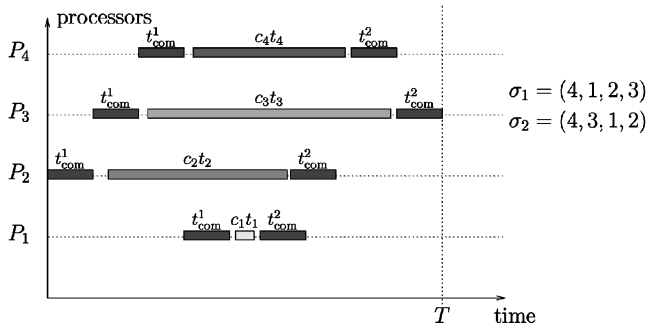

Fig. 2. Delaying messages sent back to the host.

MaxTasks1(T). Given a time bound T, determine the best allocation order of tasks to slaves, i.e., a permutation σ and an allocation $\mathcal{C} = \{c_1, c_2, \ldots, c_p\}$ s.t. all processors complete their execution within T units of time and the total number of tasks is maximized:

$$\max \left(\sum_{i=1}^{p} c_i \mid \sigma \text{ permutation and } \forall i \in [1,p] : \sigma(i)t_{\text{com}} + c_i t_i \leq T \right).$$

To solve MinTime(C) in this context, we would compute the execution time on the fastest processor as $t_{\text{com}} + C \min_{1 \leq i \leq p} t_i$, and round this value up to the next power of 2: This leads to the value of T used in the dichotomic search.

2.3 With Initial and Final Communications

As pointed out above, it is natural to assume that, after the processing of their tasks, slave processors will send some data back to the master. Because this message may well have a different size than the message initially sent by the master, we model this situation by using two communication costs, t^1_{com} for the messages sent by the master to the slaves, and t^2_{com} for the messages sent by the slaves to the master.

As shown above, we look for a permutation σ_1 which determines the ordering of the initial messages from the host: The host sends data to slave P_i at time $\sigma_1(i)t^1_{\text{com}}$. But, we also look for a second permutation σ_2 which determines the ordering of the final messages sent back to the host: Given a time bound T, slave P_i sends data back to the host at time $T - \sigma_2(i)t^2_{\text{com}}$. Here, σ_2 corresponds to the reverse order of the communications back to the host (see Fig. 2). This formulation is without any loss of generality: Some slave processor P_i might send its message earlier than this bound, but we can always shift the communication pattern as stated, i.e., delay some messages, as illustrated in Fig. 2. We are ready to state the optimization problem analytically:

MaxTasks2(T). Given a time bound T, determine the best allocation of tasks to slaves, i.e., two permutations σ_1 and σ_2, and an allocation $\mathcal{C} = \{c_1, c_2, \ldots, c_p\}$ s.t. all processors complete their execution within T units of time and the total number of tasks is maximized:

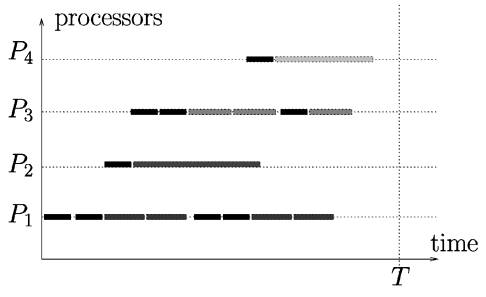

Fig. 3. Grouping some messages sent by the host to a given processor.

$$\max\left(\sum_{i=1}^{p} c_i \mid \sigma_1, \sigma_2 \text{ permutations and } \forall i \in [1,p] : \sigma_1(i)t^1_{\text{com}} + c_i t_i + \sigma_2(i)t^2_{\text{com}} \leq T\right).$$

2.4 With Communications Before Each Task Processing

We also consider the case when communications are required between the master and the slave before the processing of each task. In this third model, we consider that the cost of this communication is t_{com}. This model is quite natural: Some specific input data may well have to be propagated from the master to the slave before computation can start.

We look for three functions $f_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc}: $f_{\text{startcomm}}(i)$ represents the time-step at which the communication required by task i will begin; $f_{\text{startcomp}}(i)$ represents the time-step at which the processing of task i will begin on processor $f_{\text{proc}}(i)$. The functions $f_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc} must fulfill the following conditions:

- $\forall i \geq 1,\ \ f_{\text{startcomm}}(i+1) - f_{\text{startcomm}}(i) \geq t_{\text{com}}$, which states that communications take place in exclusive mode.
- $\forall i \geq i,\ \ f_{\text{startcomp}}(i) \geq f_{\text{startcomm}}(i) + t_{\text{com}}$, which states that the processing of task i cannot start before the end of the communication required by task i.
- $\forall 1 \leq i < j$, if $f_{\text{proc}}(i) = f_{\text{proc}}(j) = k$, then $f_{\text{startcomp}}(j) \geq f_{\text{startcomp}}(i) + t_k$, which states that tasks are processed sequentially on each processor k.
- $\forall 1 \leq i < j,\ \ \text{if } f_{\text{proc}}(i) = f_{\text{proc}}(j) = k,$, then

$$[f_{\text{startcomm}}(j), f_{\text{startcomm}}(j) + t_{\text{com}}] \cap [f_{\text{startcomp}}(i), f_{\text{startcomp}}(i) + t_k] = \emptyset,$$

  which states that communications and computations cannot be overlapped on processor k.

This formulation is quite general. Note that each processor can perform several communications before processing the corresponding tasks, as illustrated in Fig. 3. We are ready to state the optimization problem analytically:

MaxTasks3(T). Given a time bound T, determine the best allocation of tasks to slaves, i.e., three functions $f_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc}, satisfying all the conditions stated above, s.t. all processors complete their execution within T units of time and the total number of tasks is maximized:

$$\max\big(N \mid \forall i \leq N,\ f_{\text{startcomp}}(i) + t_{f_{\text{proc}}(i)} \leq T\big).$$

2.5 With Communications Both Before and After Each Task Processing

It is natural to assume that, after the processing of each task, slave processors will send some data back to the master. As seen previously, we model this situation by using two different communication costs, t^1_{com} for the messages sent by the master to the slaves, and t^2_{com} for the messages sent by the slaves to the master.

We look for four functions: $f^1_{\text{startcomm}}$, $f^2_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc}: $f^1_{\text{startcomm}}(i)$ represents the time-step at which the communication from the host required before task i will begin (just as $f_{\text{startcomm}}(i)$ in the previous section); similarly, $f^2_{\text{startcomm}}(i)$ represents the time-step at which the communication back to the host after task i will begin. Finally, $f_{\text{startcomp}}$ and $f_{\text{proc}}(i)$ are defined as before: $f_{\text{startcomp}}(i)$ represents the time-step at which the processing of task i will begin on processor $f_{\text{proc}}(i)$. The functions $f^1_{\text{startcomm}}$, $f^2_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc} have to fulfill the following conditions:

- $\forall i \geq 1,\ \forall j \geq 1,\ \ \forall (k,l) \in \{1,2\},\ \text{if } k \neq l \text{ or } i \neq j$ then

$$[f^k_{\text{startcomm}}(i), f^k_{\text{startcomm}}(i) + t^k_{\text{com}}] \cap [f^l_{\text{startcomm}}(j), f^l_{\text{startcomm}}(j) + t^j_{\text{com}}] = \emptyset,$$

  which states that communications take place in exclusive mode.
- $\forall i \geq 1,\ \ f_{\text{startcomp}}(i) \geq f_{\text{startcomm}}(i) + t^1_{\text{com}}$, which states that the processing of task i cannot start before the end of the communication from the host required by task i.
- $\forall i \geq 1, f_{\text{startcomp}}(i) + t_{f_{\text{proc}}(i)} \leq f^2_{\text{startcomm}}(i)$, which states that the communication back to the host required after task i cannot start before the end of the processing of task i.
- $\forall 1 \leq i < j,\ \text{if } f_{\text{proc}}(i) = f_{\text{proc}}(j) = k,\ f_{\text{startcomp}}(j) \geq f_{\text{startcomp}}(i) + t_k$, which states that tasks are processed sequentially on each processor k.
- $\forall 1 \leq i < j$, if $f_{\text{proc}}(i) = f_{\text{proc}}(j) = k$, then

$$[f^1_{\text{startcomm}}(j), f^1_{\text{startcomm}}(j) + t_{\text{com}}] \cap [f_{\text{startcomp}}(i), f_{\text{startcomp}}(i) + t_k] = \emptyset$$

  and

$$[f^2_{\text{startcomm}}(i), f^2_{\text{startcomm}}(i) + t_{\text{com}}] \cap [f_{\text{startcomp}}(j), f_{\text{startcomp}}(j) + t_k] = \emptyset,$$

  which states that communications and computations cannot be overlapped on processor k.

Again, this formulation is quite general. Note that each processor can perform several communications from the host before processing the corresponding tasks, as well as delaying several communications back to the host: this is

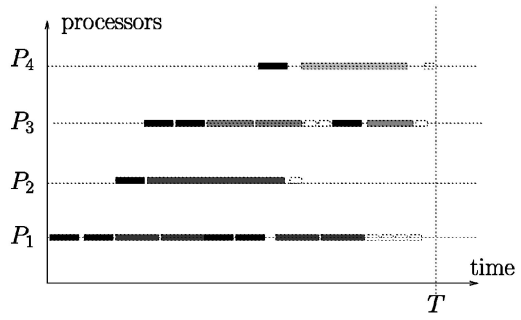

Fig. 4. Grouping some messages sent by and to the host.

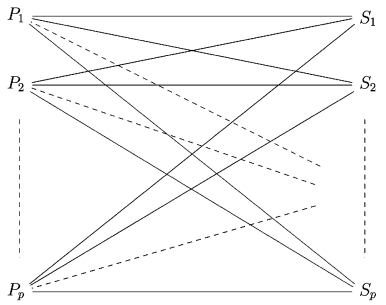

Fig. 5. Bipartite graph for MaxTasks1(T).

illustrated in Fig. 4. We are ready to state the optimization problem analytically:

MaxTasks4(T). Given a time bound T, determine the best allocation of tasks to slaves, i.e., four functions $f^1_{\text{startcomm}}$, $f^2_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc} satisfying all the conditions stated above, s.t. all processors complete their execution within T units of time and the total number of tasks is maximized:

$$\max\bigl(N \mid \forall i \leq N,\ f^2_{\text{startcomm}}(i) + t^2_{\text{com}} \leq T\bigr).$$

3 Solution with an Initial Scattering of Data

3.1 Restricted Search

To (partially) solve the MaxTasks1(T) problem of Section 2.2, Andonie et al. [1] restrict the search to allocations where the fastest processors start computing first. They use a dynamic programming algorithm to solve the optimization problem MinTime(C). With our setting for problem MaxTasks1(T), this amounts to sorting the cycle-times as $t_1 \leq t_2 \leq \ldots \leq t_p$, and to letting $\sigma(i) = i$ for $1 \leq i \leq p$. The intuition is that since fastest processors execute tasks more rapidly than the others, they should work longer.

However, the intuition is misleading in some cases. Assume, for instance, two slave processors ($p = 2$) with $t_1 = 5$ and $t_2 = 9$, and let $t_{\text{com}} = 1$. For the time bound $T = 28$, it is better to start the slow processor P_2 first: P_2 can then execute three tasks: $t_{\text{com}} + 3t_2 = 28 \leq T$; the fast processor, although started at time-step $2t_{\text{com}} = 2$, can execute five tasks: $2t_{\text{com}} + 5t_1 = 27 \leq T$. If we start the fast processor first, it cannot execute more than five tasks, while the second processor can execute only two.

3.2 Matching Techniques

The optimal solution to the MaxTasks1(T) problem can be found using a weighted-matching algorithm. The idea is to draw a complete bipartite graph with $2p$ vertices, as shown in Fig. 5. Vertices on the left represent processors, while vertices on the right represent possible values for the permutation σ. The edge from vertex P_i to vertex S_j is weighted with the maximum number of tasks that P_i can execute if $\sigma(i) = j$, namely, $\left\lfloor \frac{T - jt_{\text{com}}}{t_i} \right\rfloor$. It is easy to see that there is a one-to-one correspondence between matchings in this complete bipartite graph and permutations. Because the total weight of a given matching is the total number of tasks that can be executed for the corresponding choice of the permutation, our problem reduces to finding the maximum weighted matching in the bipartite graph. Efficient (polynomial) algorithms exist to solve this problem, see [12], [22]. More precisely, the complexity of finding a maximum weighted matching in a bipartite graph with $2p$ vertices is of order $O(p^{\frac{5}{2}})$ [15]. We formally state our result:

Proposition 1. *An optimal solution to the MaxTasks1(T) problem with p processors can be found in $O(p^{\frac{5}{2}})$ time using a maximum weighted matching algorithm.*

Note that the overhead induced by the search for an optimal task allocation with the matching algorithm does depend on p, the number of processors, but not on $\sum_{i=1}^{p} c_i$, the total number of tasks to be processed. Therefore, this overhead becomes negligible when the number of tasks becomes large.

To conclude this section, we work out the example given in Section 3.1: With two processors, $t_1 = 5$, $t_2 = 9$, $t_{\text{com}} = 1$, and $T = 28$, we obtain the weighted graph of Fig. 6. For instance, the weight of the edge (P_1, S_2) is $\left\lfloor \frac{28 - 2t_{\text{com}}}{t_1} \right\rfloor = 5$. The maximum weighted matching is composed of the two edges, (P_1, S_2) and (P_2, S_1), and has total weight 8, which is indeed the maximum number of tasks that can be executed within 28 time-steps.

4 Solution with Initial and Final Communications

The solution to the MaxTasks2(T) problem with initial and final messages turns out to be surprisingly difficult. In fact, we prove that this problem is strongly NP-Complete in Section 4.1. Nevertheless, we present an efficient guaranteed approximation using matching techniques, as explained in Section 4.2.

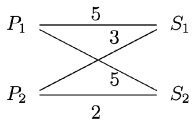

Fig. 6. Working out the example of Section 3.1.

4.1 Strong NP-Completeness of MaxTasks2(*T*)

In this section, we prove the strong NP-Completeness of MaxTasks2(T). The decision problem associated to MaxTasks2(T) is the following:

MaxTasks2-Dec(T, K). Given a time bound T and a bound K, determine an allocation order of tasks to slave, i.e., two permutations σ_1 and σ_2, and an allocation $\mathcal{C} = \{c_1, c_2, \ldots, c_p\}$ s.t. all processors complete their execution within T units of time and the total number of tasks is at least K:

$$\text{Find } \sigma_1, \sigma_2 \text{ permutations}, c_1, c_2, \ldots, c_p, \sum c_i = K,$$
$$\forall i \in [1,p] : \sigma_1(i)t^1_{\text{com}} + c_i t_i + \sigma_2(i)t^2_{\text{com}} \leq T.$$

In order to prove the strong NP-Completeness of MaxTasks2-Dec(T, K), we use a reduction to RN-3DM, a special instance of Numerical 3-Dimensional Matching that has been proved to be strongly NP-Complete by Yu [23]. A general instance of RN-3DM is as follows:

RN-3DM: Let $U = (u_1, u_2, \ldots, u_p)$ and e such that $\sum_{i=1}^{p} u_i = pe + p(p+1)$. Are there two permutations λ_1 and λ_2, such that

$$\forall 1 \leq i \leq p, \quad \lambda_1(i) + \lambda_2(i) + u_i = e?$$

We can now prove the following theorem:

Theorem 1. *MaxTasks2-Dec(T, K) is strongly NP-Complete.*

Proof. We consider the following instance of MaxTasks2-Dec(T, K). Let $t^1_{\text{com}} = t^2_{\text{com}} = 1$, $m = \max_{1 \leq i \leq p} u_i$, $t_i = u_i + e + m$ for $1 \leq i \leq p$, $T = 2e + m$, and $K = p$. Clearly, the size of this instance of MaxTasks2-Dec(T, K) is polynomial (and even linear) in the size of the original instance of RN-3DM. We prove that the above instance of MaxTasks2-Dec(T, K) has a solution if and only if the general instance of RN-3DM has a solution.

Let us first suppose that the general instance of RN-3DM has a solution. Then, there exist two permutations, λ_1 and λ_2, such that

$$\forall 1 \leq i \leq p, \quad \lambda_1(i) + \lambda_2(i) + u_i = e.$$

We let $\sigma_1 = \lambda_1$ and $\sigma_2 = \lambda_2$. Then,

$$\begin{aligned} \forall 1 \leq i \leq p, \quad \lambda_1(i) + \lambda_2(i) + u_i = e \quad &\Longrightarrow \quad \sigma_1(i) + u_i + e + \\ &\quad m + \sigma_2(i) = 2e + m \\ &\Longrightarrow \quad \sigma_1(i) + t_i + \sigma_2(i) \\ &\quad = T. \end{aligned}$$

Therefore, each slave can perform exactly one task in time T, and we have built a solution to our instance of MaxTasks2-Dec(T, K). Reciprocally, suppose that we have built a solution to our instance of MaxTasks2-Dec(T, K), i.e., two permutations σ_1 and σ_2, and an allocation $\mathcal{C} = \{c_1, c_2, \ldots, c_p\}$ s.t. $\sum_{i=1}^{p} c_i = p$ and

$$\forall i \in [1,p] : \sigma_1(i) + c_i t_i + \sigma_2(i) \leq T.$$

Then,

$$\forall 1 \leq i \leq p, \quad \sigma_1(i) + \sigma_2(i) + c_i(u_i + e + m) \leq 2e + \max u_i.$$

First, let us remark that necessarily, $\forall i,\ c_i < 2$. Indeed, if $\exists i,\ c_i \geq 2$, then

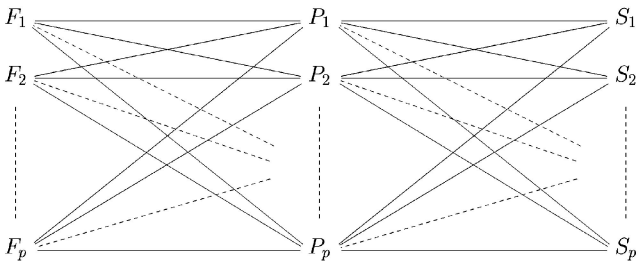

Fig. 7. Bipartite graph with initial and final communications.

$$\begin{aligned} c_i(u_i + e + \max u_i) \quad &\geq \quad 2u_i + 2e + 2\max u_i \\ &> \quad 2e + max\ u_i = T, \end{aligned}$$

which is clearly impossible since the tasks assigned to processor i would not be processed within T units of time. Thus, since $\sum_{i=1}^{p} c_i = p$ and $\forall i,\ c_i < 2$, then $\forall i,\ c_i = 1$. Therefore, if we set $\lambda_1 = \sigma_1$ and $\lambda_2 = \sigma_2$, then

$$\forall 1 \leq i \leq p, \ \ \sigma_1(i) + u_i + e + m + \sigma_2(i) \leq 2e + m \Longrightarrow$$
$$\lambda_1(i) + \lambda_2(i) + u_i \leq e.$$

By summing all the inequalities above, we obtain $p(p+1) + \sum_{i=1}^{p} u_i \leq pe$ and, since $p(p+1) + \sum_{i=1}^{p} u_i = pe$, all the inequalities above are in fact tight. Hence, we have built a solution to the general instance of RN-3DM.

This achieves the proof of the strong NP-Completeness of MaxTasks2-Dec(T, K). □

4.2 Matching Techniques

To take both permutations σ_1 and σ_2 into account, we build a bipartite graph $G = (V, E)$ with $3p$ vertices (i.e., $|V| = 3p$), as shown Fig. 7. The p leftmost vertices F_i correspond to the first permutation σ_1, the p center vertices P_i correspond to processors, and the p rightmost vertices S_i correspond to the second permutation σ_2. Rather than a matching, we extract from the graph a subset E' of $2p$ edges so that, in the graph $G = (V, E')$, each vertex F_i or S_i is exactly of degree 1, and each vertex P_i is exactly of degree 2. The complexity of extracting such a spanning subgraph from the graph with $3p$ vertices is of order $O(p^{\frac{5}{2}})$ again since we can solve independently the maximum weighted matching in both bipartite graphs with $2p$ vertices (on the left-hand size and on the right-hand size in Fig. 7) in time of order $O(p^{\frac{5}{2}})$.

The problem is that edge weights cannot be determined fully accurately; the inequality $\sigma_1(i)t^1_{\text{com}} + c_i t_i + \sigma_2(i)t^2_{\text{com}} \leq T$ translates into

$$c_i \leq \left\lfloor \frac{T - \sigma_1(i)t^1_{\text{com}} - \sigma_2(i)t^2_{\text{com}}}{t_i} \right\rfloor,$$

and we need to know *both* $\sigma_1(i)$ and $\sigma_2(i)$ to compute c_i. Instead, we use the approximation

$$\left\lfloor \frac{T/2 - \sigma_1(i)t^1_{\text{com}}}{t_i} \right\rfloor + \left\lfloor \frac{T/2 - \sigma_2(i)t^2_{\text{com}}}{t_i} \right\rfloor.$$

This approximation enables us to weight the edges as follows: The edge between F_j and P_i is weighted as

$$\left\lfloor \frac{T/2 - jt^1_{\text{com}}}{t_i} \right\rfloor,$$

while the edge between P_i and S_k is weighted as

$$\left\lfloor \frac{T/2 - kt^2_{\text{com}}}{t_i} \right\rfloor.$$

Theorem 2. *The previous approximation leads to tasks allocations that differs at most by p from the optimal solution.*

Proof. First, note that

$$\forall a, b : 0 \leq \lfloor a+b \rfloor - \lfloor a \rfloor - \lfloor b \rfloor \leq 1. \quad (1)$$

Therefore, for any allocation (σ_1, σ_2, c) built using the previous approximation, we have

$$\begin{aligned} \forall i,\ c_i &\leq \left\lfloor \frac{T/2 - \sigma_1(i)t^1_{\text{com}}}{t_i} \right\rfloor + \left\lfloor \frac{T/2 - \sigma_2(i)t^2_{\text{com}}}{t_i} \right\rfloor \\ &\leq \left\lfloor \frac{T - \sigma_1(i)t^1_{\text{com}} - \sigma_2(i)t^2_{\text{com}}}{t_i} \right\rfloor, \end{aligned}$$

and (σ_1, σ_2, c) is then a solution of the initial problem.

Let us note by

$$C_{app}(\sigma_1, \sigma_2) = \sum_{i=1}^{p} \left\lfloor \frac{T/2 - \sigma_1(i)t^1_{\text{com}}}{t_i} \right\rfloor + \left\lfloor \frac{T/2 - \sigma_2(i)t^2_{\text{com}}}{t_i} \right\rfloor,$$

the approximative cost of two permutations and

$$\sigma_{app} = (\sigma_1^{(app)}, \sigma_2^{(app)}) = \underset{\sigma_1, \sigma_2}{\operatorname{argmax}}\, C_{app}(\sigma_1, \sigma_2),$$

the permutations built by our algorithm.

Let

$$C_{opt}(\sigma_1, \sigma_2) = \sum_{i=1}^{p} \left\lfloor \frac{T - \sigma_1(i)t^1_{\text{com}} - \sigma_2(i)t^2_{\text{com}}}{t_i} \right\rfloor$$

denote the real cost of two permutations and let

$$\sigma_{opt}(\sigma_1^{(opt)}, \sigma_2^{(opt)}) = \underset{\sigma_1, \sigma_2}{\operatorname{argmax}}\, C_{opt}(\sigma_1, \sigma_2)$$

be some permutations that are optimal solutions to the original problem.

By definition,

$$C_{opt}(\sigma_{app}) \leq C_{opt}(\sigma_{opt}) \quad (2)$$

and

$$C_{app}(\sigma_{opt}) \leq C_{app}(\sigma_{app}). \quad (3)$$

Using (1), we find that

$$\forall \sigma : C_{opt}(\sigma) - p \leq C_{app}(\sigma) \leq C_{opt}(\sigma) \quad (4)$$

and, therefore,

$$\begin{aligned} C_{opt}(\sigma_{opt}) - p &\leq C_{app}(\sigma_{opt}) && \text{(using (4))} \\ &\leq C_{app}(\sigma_{app}) && \text{(using (3))} \\ &\leq C_{opt}(\sigma_{app}) && \text{(using (4))} \\ C_{opt}(\sigma_{opt}) - p &\leq C_{opt}(\sigma_{app}) \leq C_{opt}(\sigma_{opt}) && \text{(using (2)),} \end{aligned}$$

which means that our approximation leads to tasks allocations that differs at most by p from the optimal solution. □

4.3 Simulations

In this section, we compare solutions given by the approximation algorithm proposed in Section 4, with the optimal ones. For a given instance of the problem, we compute the difference between number of tasks processed using the optimal permutations (found with an exhaustive search), and the number of tasks processed using the approximation algorithm proposed in Section 4. Fig. 8 depicts the mean value of this error for several tests when the number of processors varies:

1. In Fig. 8a, cycle times are moderately heterogeneous (numbers at random between 15 and 25) and communication times can be large compared to cycle times (numbers at random between 10 and 90).
2. In Fig. 8b, cycle times are moderately heterogeneous (numbers at random between 15 and 25), but communication times are rather small compared to cycle times (between 1 and 9).
3. In Fig. 8c, cycle times are strongly heterogeneous (numbers at random between between 5 and 35) and communication times can be large compared to cycle times (two numbers at random between 10 and 90).
4. In Fig. 8d, cycle times are strongly heterogeneous (numbers at random between between 5 and 35) and communication times are rather small compared to cycle times (two numbers at random between 1 and 9).

In all cases, the matching gives very satisfying results since the difference between the optimal number of tasks that may be processed and the number of tasks processed using the matching algorithm is quite small. As we could expect, this difference grows with p, but does not strongly depend upon the degree of heterogeneity.

5 Solution with Communications Before Each Task

In this section, we present an asymptotically optimal algorithm for MaxTasks3(T): When T becomes large, the ratio of the number of tasks processed by this algorithm over the number of tasks executed by the optimal solution tends to one.

5.1 Theoretical Bounds

In order to prove the asymptotic optimality of our algorithm, we need to determine the optimal number of tasks that can be performed if the cost of a communication between the master and the slave is t_{com} and the cycle times of slaves processors are $t_1 \leq t_2 \leq \ldots \leq t_p$. Consider a valid communication and computation scheme, i.e., three functions $f_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc}, satisfying the conditions given in Section 2.4. Let T be the time bound and let N denote the maximal number of tasks that can be processed within T units of time:

$$N = \max\{n,\ \forall i \leq n,\ f_{\text{startcomp}}(i) + T_{f_{\text{proc}}(i)} \leq T\}.$$

Moreover, let

$$\forall i,\ 1 \leq i \leq p,\ \ \alpha_i = \frac{\text{Card}\{j,\ \ f_{\text{proc}}(j) = i\}\ \ t_{\text{com}}}{T}$$

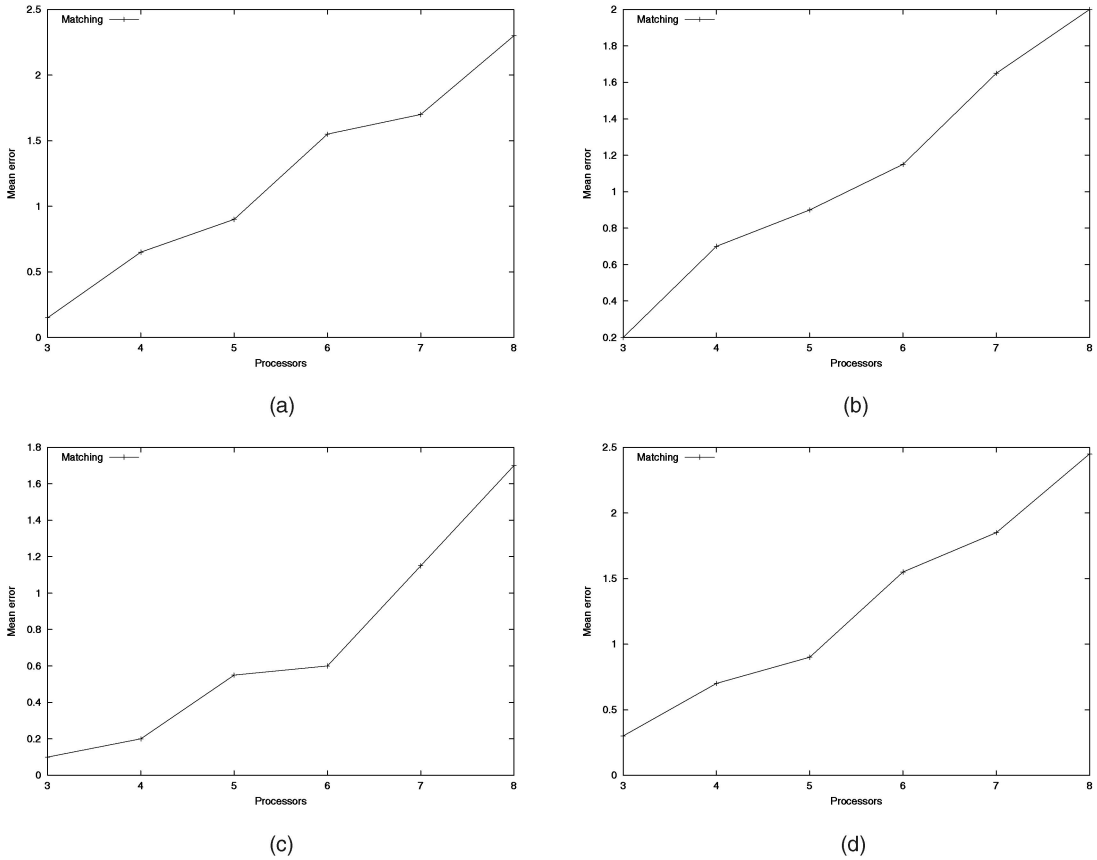

Fig. 8. A few simulations. (a) comp 20 ± 5; comm 50 ± 40, (b) comp 20 ± 5; comm 5 ± 4, (c) comp 20 ± 15; comm 50 ± 40, and (d) comp 20 ± 15; comm 5 ± 4.

be the ratio of the time spent by the master to perform communications with slave P_i over the time bound T.

Lemma 1. *With the above notations, we have*

$$N \leq \sum_i \frac{T}{t_{\text{com}} + t_i} \tag{5}$$

$$N \leq \frac{T}{t_{\text{com}}} \tag{6}$$

Proof.

1. Since $\forall 1 \leq i \leq p,\ \alpha_i = \frac{\text{Card}\{j,\ f_{\text{proc}}(j)=i\}\ t_{\text{com}}}{T}$, then $\frac{T\alpha_i}{t_{\text{com}}} = \text{Card}\{j,\ f_{\text{proc}}(j) = i\}$ and
$$\frac{T\sum_{i=1}^{p} \alpha_i}{t_{\text{com}}} = \sum_{i=1}^{p} \text{Card}\{j,\ f_{\text{proc}}(j) = i\} = N. \tag{7}$$
2. Let us determine the maximal number of tasks that can be performed by slave P_i, $\forall 1 \leq i \leq p$. Since the overall cost of a task on slave P_i is $t_{\text{com}} + t_i$, we have
$$\text{Card}\{j,\ f_{\text{proc}}(j) = i\} \leq \frac{T}{t_{\text{com}} + t_i},\ \text{i.e.,}\ \alpha_i \leq \frac{t_{\text{com}}}{t_{\text{com}} + t_i}. \tag{8}$$
3. Let us determine the number of communications that can be performed by the master. The cost of a communication is t_{com}, so that $Nt_{\text{com}} \leq T$ and, then, using (7), we deduce
$$\sum_{i=1}^{p} \alpha_i \leq 1. \tag{9}$$
4. Using (7) and (8), we have
$$N \leq \sum_i \frac{T}{t_{\text{com}} + t_i}.$$
5. Using (7) and (9), we deduce
$$N \leq \frac{T}{t_{\text{com}}}.$$

□

In order to determine the optimal number of tasks that can be performed during T time steps, we need to distinguish two different cases, according to the value of $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i}$. Indeed, it turns out that the communication network is not the limiting resource if $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} \leq 1$, but it becomes the limiting resource otherwise.

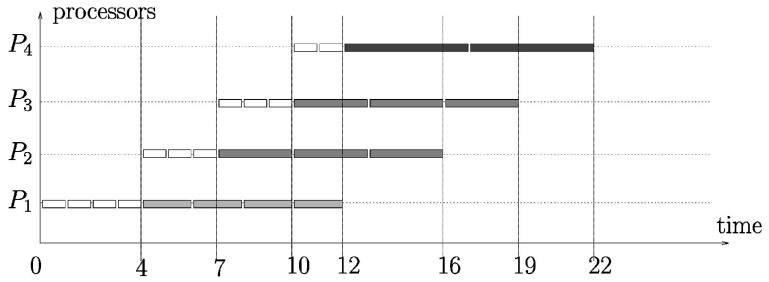

Fig. 9. Example when $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} \leq 1$.

5.2 Solution of MaxTasks3(*T*) if $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} \leq 1$

To solve MaxTasks3(T), we propose an algorithm that consists in determining a pattern for communications and computations, that will be reproduced periodically throughout the execution.

Let $tc_i = t_{\text{com}} + t_i$, for $1 \leq i \leq p$, denote the overall cost of the processing of a task on slave P_i since we cannot overlap communications and computations. Let T^{pattern} be the least common multiple of these p values tc_i: T^{pattern} determines the length of the pattern. Let $nb_i^{\text{pattern}} = \frac{T^{\text{pattern}}}{tc_i}$ be the number of tasks processed by processor P_i during the execution of the pattern.

Consider the following example: $t_{\text{com}} = 1$ and $p = 4$ slave processors with $t_1 = 2$, $t_2 = 3$, $t_3 = 3$, and $t_4 = 5$. We have $tc_1 = 3$, $tc_2 = 4$, $tc_3 = 4$, and $tc_4 = 6$. In this case, $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} = 1$, $T^{\text{pattern}} = 12$, and $nb_1^{\text{pattern}} = 4$, $nb_2^{\text{pattern}} = 3$, $nb_3^{\text{pattern}} = 3$, and $nb_4^{\text{pattern}} = 2$.

To formally build the pattern, we need some complicated notations. We advise the reader to follow the construction for the example, using Fig. 9. First, we define time-steps and processors within the pattern, using three new functions $f_{\text{startcomm}}^{\text{pattern}}$, $f_{\text{startcomp}}^{\text{pattern}}$, and $f_{\text{proc}}^{\text{pattern}}$, which we define as follows (initially, $nb_0^{\text{pattern}} = 0$):

- Determine which processor executes task number i:

$$\forall 1 \leq i \leq \sum_{k=1}^{p} nb_k^{\text{pattern}} : f_{\text{proc}}^{\text{pattern}}(i) = \min\left\{ j \mid i > \sum_{k=0}^{j-1} nb_k^{\text{pattern}} \right\}.$$

- Determine the beginning of the communication and of the computation for task number i:

$$f_{\text{startcomm}}^{\text{pattern}}(i) = 1 + \sum_{k=0}^{j-1} nb_k^{\text{pattern}}(t_{\text{com}} + t_k) + (i - 1 - \sum_{k=0}^{j-1} nb_k^{\text{pattern}}) t_{\text{com}},$$

$$f_{\text{startcomp}}^{\text{pattern}}(i) = 1 + \sum_{k=0}^{j} nb_k^{\text{pattern}} t_{\text{com}} + \sum_{k=0}^{j-1} nb_k^{\text{pattern}} t_k + (i - 1 - \sum_{k=0}^{j-1} nb_k^{\text{pattern}}) t_j.$$

The important fact about this pattern is that $\forall 2 \leq i \leq p$,

$$\left(f_{\text{startcomp}}^{\text{pattern}}\left(\sum_{k=0}^{i} nb_k^{\text{pattern}} \right) + t_i \right) - \left(f_{\text{startcomp}}^{\text{pattern}}\left(\sum_{k=0}^{i-1} nb_k^{\text{pattern}} \right) + t_{i-1} \right) = \left(f_{\text{startcomm}}^{\text{pattern}}\left(\sum_{k=0}^{i-1} nb_k^{\text{pattern}} + 1 \right) \right) - \left(f_{\text{startcomm}}^{\text{pattern}}\left(\sum_{k=0}^{i-2} nb_k^{\text{pattern}} + 1 \right) \right).$$

This condition states that the difference between the date of the end of the processing of the last task on slave P_i and the date of the end of the processing of the last task on slave P_{i-1} is equal to the difference between the beginning of the communication required by the first task processed by slave P_i and the beginning of the communication required by the first task processed by processor P_{i-1}. This is the key-condition that ensures the periodicity of the whole computation and communication scheme from one pattern to the next one: It is possible to execute the pattern defined above every T^{pattern} time steps, as illustrated in Fig. 9. During the execution of one pattern, we process $\sum_{i=1}^{p} nb_k^{\text{pattern}}$ tasks.

We are now able to define the functions $f_{\text{startcomm}}(n)$, $f_{\text{startcomp}}(n)$, and $f_{\text{proc}}(n)$ corresponding to our algorithm. Let k s.t. $k \sum_{i=1}^{p} nb_i^{\text{pattern}} < n \leq (k+1) \sum_{i=1}^{p} nb_i^{\text{pattern}}$ and let $i = n - k \sum_{i=1}^{p} nb_i^{\text{pattern}}$. Then,

$$f_{\text{startcomm}}(n) = kT^{\text{pattern}} + f_{\text{startcomm}}^{\text{pattern}}(i),$$
$$f_{\text{startcomp}}(n) = kT^{\text{pattern}} + f_{\text{startcomp}}^{\text{pattern}}(i),$$
$$f_{\text{proc}}(n) = f_{\text{proc}}^{\text{pattern}}(i).$$

One can easily check that the functions $f_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc} satisfy all the conditions stated in Section 2.4.

5.3 Solution of MaxTasks3(*T*) if $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} > 1$

To solve MaxTasks3(T) when $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} > 1$, we slightly modify the algorithm proposed in the previous section. Indeed, in this case, the network is the limiting resource. The algorithm consists of determining a communication and computation pattern so that the communication network is always in use. Some slower processors will be kept idle at some periods, or even will never be used.

First of all, we sort the cycle-times of the slave processors and assume that

$$t_1 \leq t_2 \leq \ldots \leq t_p.$$

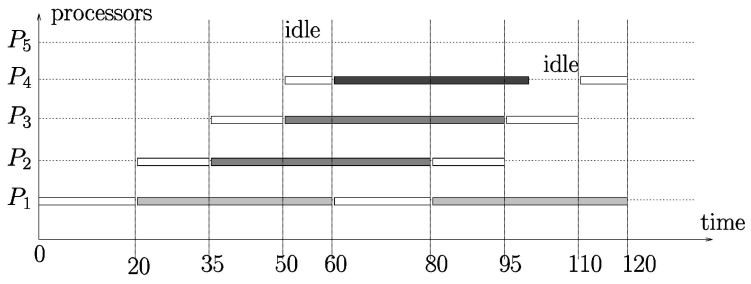

Fig. 10. Example when $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} > 1$.

Let tc_i be defined as previously and let

$$p_{max} = \max\left\{k \mid \sum_{i=1}^{k} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} \leq 1\right\}.$$

p_{max} is the index of the last processor whose computation power will be fully used in the pattern.

Let T^{pattern} be the least common multiple of t_{com} and of the tc_i, $1 \leq i \leq p_{max}$. Moreover, define nb_i^{pattern} as follows:

$$\forall\, 1 \leq i \leq p_{max}, \quad nb_i^{\text{pattern}} = \frac{T^{\text{pattern}}}{tc_i},$$

$$nb_{p_{max}+1}^{\text{pattern}} = \frac{T^{\text{pattern}} - t_{\text{com}} \sum_{i=1}^{p_{max}} nb_i^{\text{pattern}}}{t_{\text{com}}},$$

and let

$$nb_i^{\text{pattern}} = 0, \quad \forall\, i > p_{max}+1.$$

We see that processor number $p_{max}+1$ is not fully used, while other processors are not used at all.

With these notations, we define $f_{\text{startcomm}}^{\text{pattern}}$, $f_{\text{startcomp}}^{\text{pattern}}$, and $f_{\text{proc}}^{\text{pattern}}$ as in the previous section. Again, the only difference is that slaves P_i, $i > p_{max}+1$ are kept idle all the time, while slave $P_{p_{max}+1}$ is kept idle during the last $\left(T^{\text{pattern}} - nb_{p_{max}+1}^{\text{pattern}} tc_{p_{max}+1}\right)$ time steps.

The construction of the pattern is illustrated in Fig. 10, with $t_{\text{com}} = 1$, and $p = 5$ processors s.t. $t_1 = 2$, $t_2 = 3$, $t_3 = 3$, $t_4 = 4$, and $t_5 = 6$. In this case, $\sum_i \frac{t_{\text{com}}}{t_{\text{com}}+t_i} > 1$, $p_{max} = 3$, $T^{\text{pattern}} = 60$, and $nb_1^{\text{pattern}} = 20$, $nb_2^{\text{pattern}} = 15$, $nb_3^{\text{pattern}} = 15$, $nb_4^{\text{pattern}} = \frac{60-(20+15+15)*1}{1} = 10$, and $nb_5^{\text{pattern}} = 0$. Slave P_5 is not used at all, while slave P_4 stays idle the last $(60 - 10 * 5) = 10$ time-steps of the pattern.

We extend the definition of $f_{\text{startcomm}}^{\text{pattern}}$, $f_{\text{startcomp}}^{\text{pattern}}$, and $f_{\text{proc}}^{\text{pattern}}$ to $f_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc} exactly as before. Again, one can easily check that the functions $f_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc} satisfy all the conditions stated in Section 2.4.

5.4 Asymptotic Optimality

In this section, we show that the algorithm presented in Sections 5.2 and 5.3 is asymptotically optimal.

- Consider first the case $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} \leq 1$. Let N be the number of tasks that can be performed using our algorithm. We define k such that

$$kT^{\text{pattern}} \leq T - t_{\text{com}} \sum_{i=1}^{p-1} nb_i^{\text{pattern}} < (k+1)T^{\text{pattern}}.$$

In this expression, $t_{\text{com}} \sum_{i=1}^{p-1} nb_i^{\text{pattern}}$ represents the delay before slave P_p begins to receive its first message, as illustrated in Fig. 11, and k represents the number of patterns that can be entirely completed before time bound T. Then,

$$k \geq \frac{T}{T^{\text{pattern}}} - \frac{\sum_{i=1}^{p-1} nb_i^{\text{pattern}}}{T^{\text{pattern}}} - 1.$$

Therefore,

$$\begin{aligned} N &\geq k \sum_{i=1}^{p} nb_i^{\text{pattern}} \\ &\geq k \sum_{i=1}^{p} \frac{T^{\text{pattern}}}{t_{\text{com}}+t_i} \\ &\geq \left(\frac{T}{T^{\text{pattern}}} - \frac{\sum_{i=1}^{p-1} nb_i^{\text{pattern}}}{T^{\text{pattern}}} - 1\right) \sum_{i=1}^{p} \frac{T^{\text{pattern}}}{t_{\text{com}}+t_i} \\ &\geq \left(\sum_{i=1}^{p} \frac{T}{t_{\text{com}}+t_i}\right) - \left(\sum_{i=1}^{p-1} nb_i^{\text{pattern}} + T^{\text{pattern}}\right) \\ &\quad \left(\sum_{i=1}^{p} \frac{1}{t_{\text{com}}+t_i}\right). \end{aligned}$$

Lemma 1 ((5)) states that the optimal number of tasks N_{opt} that can be performed within T units of time satisfies

$$N_{opt} \leq \sum_{i=1}^{p} \frac{T}{t_{\text{com}}+t_i}.$$

Then, we have

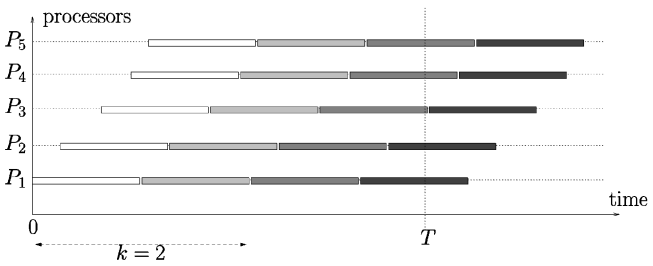

Fig. 11. Several consecutive patterns.

$$1 \geq \frac{N}{N_{opt}} \geq 1 + O\left(\frac{1}{T}\right),$$

which achieves the proof of the asymptotic optimality of our algorithm in the case $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} \leq 1$.

- Now, consider the case $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} > 1$. Let N be the number of tasks that can be performed using our algorithm. We define k such that

$$kT^{\text{pattern}} \leq T - t_{\text{com}} \sum_{i=1}^{p_{max}} nb_i^{\text{pattern}} < (k+1)T^{\text{pattern}}.$$

In this expression, $t_{\text{com}} \sum_{i=1}^{p_{max}} nb_i^{\text{pattern}}$ represents the delay before slave $P_{p_{max}+1}$ begins to receive its first message, and k represents the number of patterns that can be entirely completed before time bound T. Then,

$$k \geq \frac{T}{T^{\text{pattern}}} - \frac{t_{\text{com}} \sum_{i=1}^{p_{max}} nb_i^{\text{pattern}}}{T^{\text{pattern}}} - 1.$$

Therefore,

$$\begin{aligned} N &\geq k\left(\sum_{i=1}^{p_{max}} nb_i^{\text{pattern}} + nb_{p_{max}+1}\right) \\ &\geq k\frac{T^{\text{pattern}}}{t_{\text{com}}} \\ &\geq \left(\frac{T}{t_{\text{com}}}\right) - \left(\sum_{i=1}^{p_{max}} nb_i^{\text{pattern}} + \frac{T^{\text{pattern}}}{t_{\text{com}}}\right). \end{aligned}$$

Lemma 1 ((6)) states that the optimal number of tasks N_{opt} that can be performed within T units of time satisfies

$$N_{opt} \leq \frac{T}{t_{\text{com}}}.$$

Then, we have

$$1 \geq \frac{N}{N_{opt}} \geq 1 + O\left(\frac{1}{T}\right),$$

which achieves the proof of the asymptotic optimality of our algorithm in the case $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} > 1$.

We formally state this important result:

Theorem 3. *Let $N_{opt}(T)$ be the optimal number of tasks that can be executed within T time-steps. Let $N(T)$ be the number of tasks executed by the algorithm of Section 5.2 if $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} \leq 1$, and by the algorithm of Section 5.2 if $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} > 1$. Then,*

$$\lim_{T\to\infty} \frac{N(T)}{N_{opt}(T)} = 1.$$

5.5 Comparison with a Greedy Algorithm

In this section, we compare the results obtained with the algorithm presented in Sections 5.2 and 5.3 against the results obtained with a greedy algorithm, which works as follows: At each time step, if k slaves with respective cycle times $t_{i_1} \leq t_{i_2} \leq \ldots \leq t_{i_k}$ are waiting for a communication from the master, and if the communication network is free during the next t_{com} time steps, then a communication is performed between the master and the fastest slave P_{i_1}. In Fig. 12, we display the solution obtained with $t_{\text{com}} = 1$ and $p = 4$ slave processors with $t_1 = 2$, $t_2 = 3$, $t_3 = 3$, and $t_4 = 5$. These results are to be compared with those obtained by our algorithm (here, we have $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} = 1$), and displayed in Fig. 9.

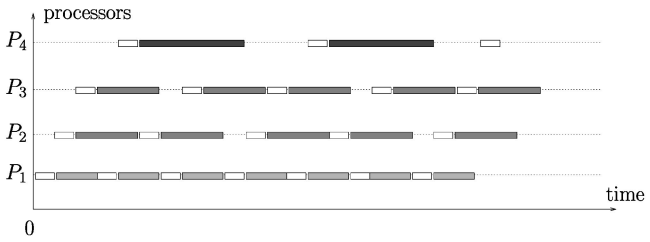

Fig. 12. Execution with the greedy algorithm.

The greedy algorithm also leads to a periodic allocation (the time period is 9); it is able to process eight tasks every nine time steps, but neither the computing resources nor the communication medium are saturated. Our algorithm is able to process 12 tasks every 12 time steps, thus leading to an improvement of order $\frac{9}{8}$.

6 Solution with Communications Both Before and After Each Task

In this section, we present an asymptotically optimal algorithm for MaxTasks4(T). The algorithm is very similar to the one presented in Section 5, so we only outline the sketch of the algorithm, and describe it through an example, without detailing the proofs.

As previously shown, we define a pattern for communications and computations that will be reproduced periodically. The pattern consists in two main phases:

- The first phase consists of both backward and forward communications between the master and the slaves.
- The second phase consists in task processing by the slaves.

Let t_{com}^1 be the communication cost for the messages from the master to the slaves, t_{com}^2 the communication cost for the messages from the slaves to the master, t_i, $1 \leq i \leq p$ the cycle times of the slaves, and T the time bound. Basically, the pattern of communications and computations is the same as those defined in Section 5, with $t_{\text{com}} = t_{\text{com}}^1 + t_{\text{com}}^2$.

The construction of the pattern is illustrated in Fig. 13, with $t_{\text{com}}^1 = 2$, $t_{\text{com}}^2 = 1$, and $p = 3$ processors: $t_1 = 8$, $t_2 = 8$, and $t_3 = 9$. In this case, $t_{\text{com}} = 3$, $\sum_i \frac{t_{\text{com}}}{t_{\text{com}}+t_i} \leq 1$, $T^{\text{pattern}} = 132$, and $nb_1^{\text{pattern}} = 12$, $nb_2^{\text{pattern}} = 12$, and $nb_3^{\text{pattern}} = 11$.

Of course, the first pattern is not executed entirely since no backward communication is required between the slaves and the master at the beginning of the execution. Similarly, the processing of tasks during the last pattern may be useless since corresponding backward messages from the slaves to the master may well not have been completed.

Nevertheless, this does not impact the asymptotic optimality of this algorithm:

Theorem 4. *Let $N_{opt}(T)$ be the optimal number of tasks that can be executed within T time-steps, and let $N(T)$ be the number of tasks executed by our algorithm. Then,*

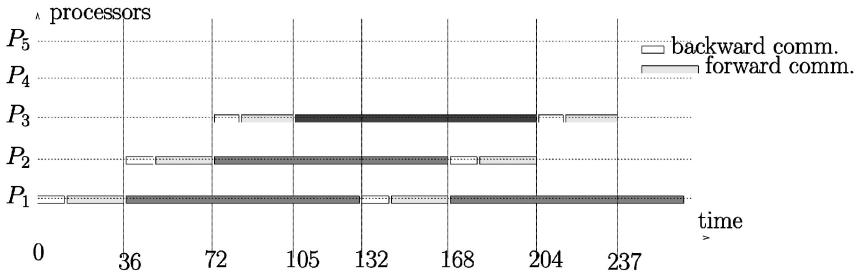

Fig. 13. Pattern when communications are required both before and after each task.

$$\lim_{T\to\infty} \frac{N(T)}{N_{opt}(T)} = 1.$$

7 Related Work

To the best of our knowledge, the most related work concerning both the application and the architecture model is presented in the paper of Andonie et al. [1], which we have already quoted in Section 3. A continuation of this work has been recently published by Chronopoulos and Jagannathan [9].

Several theoretical papers deal with complexity results for parallel machine problems with a server, establishing complexity (NP-completeness) results [13], [16], [7] and providing guaranteed approximations [17]. Before processing, each job must be loaded on a machine, which takes a certain setup time. All these setups have to be done by a single server which can handle at most one job at a time. Our first problem (with initial messages only) is a very special instance of this class of server problems.

Our second problem (with initial and final messages) is a special instance of the job-shop scheduling problem (see problem SS18 in [10]), where each job consists of only three tasks, the first and last of which having to be executed by the two machines dedicated to communications. Although this instance is very specific, we have shown that it remained NP-complete.

Generally speaking, note that our four problems differ from those cited above with a server and start-up times in that 1) all tasks are identical and independent and 2) communication times (the counterpart of the set-up times) are identical too. The difficulty lies solely in the heterogeneity of the computing resources.

Several papers by Robertazzi et al. [5], [2], [18], [8] have dealt with the distribution of a single task that can be arbitrarily partitioned in a linear fashion and then can be distributed to more than one processor to achieve better completion time. Optimal solutions have been derived when the target architecture is either a bus with heterogeneous processors (i.e., the architecture considered in this paper) or a tree. In this context, the main difference is that the cost of a communication depends on the amount of processing assigned to a processor, contrarily to the assumption made in Sections 3 and 4. Surprisingly, in this case, on a homogeneous bus, the overall completion time does not depend on the order of the communications between the master and the slaves, while it has been proven to be the main issue with our application model. The modeling of an application by the divisible load approach is also different from the one considered in Section 5. Indeed, in the context of divisible loads, one single communication is performed with each slave at the beginning, and this communication involves entirely the data needed for the entire processing. On the contrary, we allow several communications between the master and the slaves and, therefore, we can obtain a better overall completion time by enabling a fastest start, hence a better overlapping, for the whole set of processors.

In [14], Hedetniemi presents an interesting algorithm for polling in a tree network of processors that can be simplified for a bus of processors. The polling operation consists in two phases: First, the same message is sent to all the nodes by the master and, then, processors send back individual results to the master. This problem is close to the one considered in Section 4, but the polling operation does not require any processing by slave processors. Interestingly, we point out that this last operation is the clue of the complexity stated in Theorem 1; indeed, an optimal polynomial-time algorithm can be derived for polling in tree networks, whereas our problem is strongly NP-Complete.

8 Conclusion

In this paper, we have shown that deriving efficient algorithms for the master-slave paradigm, in the framework of heterogeneous computing resources and communication links used in exclusive mode, turns out to be surprisingly difficult. More precisely, we have designed an optimal polynomial algorithm in the case of an initial scattering of data and provided a guaranteed polynomial approximation algorithm in the case of initial and final communications. This last problem has been proven to be strongly NP-Complete, even on (intuitively) simple instances. Finally, we have presented asymptotically optimal algorithms for the case where each task processing must be preceded (and possibly followed) by a communication from (back to) the master.

The different variants of the master-slave problem that we have addressed in this paper seem quite representative of a large class of regular problems that exhibit a simple solution in the context of homogeneous processors, but turn out to raise several algorithmic difficulties in the context of heterogeneous resources [4], [6]. Data decomposition, scheduling heuristics, and load balancing were known to be hard problems in the context of classical parallel architectures. They become extremely difficult in the context of heterogeneous clusters, not to speak about metacomputing platforms.

REFERENCES

[1] R. Andonie, A.T. Chronopoulos, D. Grosu, and H. Galmeanu, "Distributed Backpropagation Neural Networks on a PVM Heterogeneous System," *Proc. Parallel and Distributed Computing and Systems Conf.*, pp. 555-560, 1998.

[2] S. Bataineh, T.Y. Hsiung, and T.G. Robertazzi, "Closed Form Solutions for Bus and Tree Networks of Processors Load Sharing a Divisible Job," *IEEE Trans. Computers*, vol. 43, no. 10, pp. 1184-1196, Oct. 1994.

[3] O. Beaumont, V. Boudet, A. Petitet, F. Rastello, and Y. Robert, "A Proposal for a Heterogeneous Cluster ScaLAPACK (Dense Linear Solvers)," *IEEE Trans. Computers*, vol. 50, no. 10, pp. 1052-1070, Oct. 2001.

[4] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Matrix-Matrix Multiplication on Heterogeneous Platforms," *Proc. Int'l Conf. Parallel Processing*, 2000.

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

[6] P. Boulet, J. Dongarra, F. Rastello, Y. Robert, and F. Vivien, "Algorithmic Issues on Heterogeneous Computing Platforms," *Parallel Processing Letters*, vol. 9, no. 2, pp. 197-213, 1999.

[7] P. Brucker, C. Dhaenens-Flipo, S. Knust, S.A. Kravchenko, and F. Werner, "Complexity Results for Parallel Machine Problems with a Single Server," Technical Report Reihe P, No. 219, Fachbereich Mathematik Informatik, Univ. Osnabrück, 2000.

[8] S. Charcranoon, T.G. Robertazzi, and S. Luryi, "Optimizing Computing Costs Using Divisible Load Analysis," *IEEE Trans. Computers*, vol. 49, no. 9, pp. 987-991, Sept. 2000.

[9] A.T. Chronopoulos and S. Jagannathan, "A Distributed Discrete-Time Neural Network Architecture for Pattern Allocation and Control," *Proc. IPDPS Workshop Bioinspired Solutions to Parallel Processing Problems*, 2002.

[10] M.R. Garey and D.S. Johnson, *Computers and Intractability, a Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1991.

[11] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing.* MIT Press, 1996.

[12] M. Gondran and M. Minoux, *Graphs and Algorithms.* John Wiley and Sons, 1984.

[13] N. Hall, C.N. Potts, and C. Sriskandarajah, "Parallel Machine Scheduling with a Common Server," *Discrete Applied Math.*, vol. 102, pp. 223-243, 2000.

[14] S. Hedetniemi, Open Problems in Combinatorial Optimization, World Wide Web Document, http://www.cs.clemson.edu/~hedet/algorithms.html, year?

[15] J.E. Hopcroft and R.M. Karp, "An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs," *SIAM J. Computing*, vol. 2, no. 4, pp. 225-231, 1973.

[16] S.A. Kravchenko and F. Werner, "Parallel Machine Scheduling Problems with a Single Server," *Math. Computational Modelling*, vol. 26, pp. 1-11, 1997.

[17] H. Lee and M. Guignard, "A Hybrid Bounding Procedure for the Workload Allocation Problem on Parallel Unrelated Machines with Setups," *J. Operational Research Soc.*, vol. 47, pp. 1247-1261, 1996.

[18] J. Sohn, T.G. Robertazzi, and S. Luryi, "Optimizing Computing Costs Using Divisible Load Analysis," *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 3, pp. 225-234, Mar. 1998.

[19] K. Soman, R. Fraczkiewicz, C. Mumenthaler, B. von Freyberg, and T. Schaumann and W. Braun, FANTOM—(Fast Newton-Raphson Torsion Angle Minimizer), World Wide Web Document, http://www.scsb.utmb.edu/fantom/fm_home.html, 2003.

[20] J.R. Stiles, T.M. Bartol, M.M. Salpeter, and M.M. Salpeter, "Monte Carlo Simulation of Neuromuscular Transmitter Release Using MCell, a General Simulator of Cellular Physiological Processes," *Computational Neuroscience*, pp. 279-284, 1998.

[21] J. Watts and S. Taylor, "A Practical Approach to Dynamic Load Balancing," *IEEE Trans. Parallel and Distributed Systems*, vol. 9, pp. 235-248, 1998.

[22] D.B. West, *Introduction to Graph Theory.* Prentice Hall, 1996.

[23] W. Yu, "The Two-Machine Flow Shop Problem with Delays and the One-Machine Total Tardiness Problem," PhD thesis, Technishe Univ. Eidhoven, June 1996.

Olivier Beaumont received the PhD degree from the Université de Rennes in January 1999. He is currently an associate professor in the LaBRI laboratory in Bordeaux. His main research interests are parallel algorithms on distributed memory architectures.

Arnaud Legrand is currently a PhD student in the Computer Science Laboratory LIP at ENS Lyon. He is mainly interested in parallel algorithm design for heterogeneous platforms and in scheduling.

Yves Robert received the PhD degree from the Institut National Polytechnique de Grenoble in January 1986. He is currently a full professor in the Computer Science Laboratory LIP at ENS Lyon. He is the author of four books, 80 papers published in international journals, and 100 papers published in international conferences. His main research interests are scheduling techniques and parallel algorithms for clusters and grids. He is a member of the ACM and IEEE, and serves as an associate editor of *IEEE Transactions on Parallel and Distributed Systems*.