The Master-Slave Paradigm with Heterogeneous Processors

Olivier Beaumont, Arnaud Legrand and Yves Robert
LIP, UMR CNRS–ENS Lyon–INRIA 5668
Ecole Normale Suprieure de Lyon
69364 Lyon Cedex 07, France
e-mail: Firstname.Lastname@ens-lyon.fr

Abstract

In this paper, we revisit the master-slave tasking paradigm in the context of heterogeneous processors. We assume that communications take place in exclusive mode. 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 task processing, we show that the problem is at least as difficult as a problem whose complexity is open. 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.

Key words: *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).

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 [14], reactor simulations [15] or protein conformations [13].

The rest of the paper is organized as follows. In Section 2 we state four different variants of the master-slave problem: (i) with communications only before the dispatching of the tasks, (ii) with communications both before and after the processing of the tasks, (iii) with communication before each task processing and (iv) 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 seems intrinsically more difficult and we prove in Section 4 that it is at least as difficult as a problem whose complexity is open; we also prove a guaranteed approximation algorithm for the second problem in Section 4. 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.

2 Problem statement

The target architectural platform is represented in Figure 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. At a given time-step, at most one processor can communicate with the master, either to receive data from the master or to send results back to the master.

We assume that there is a pool of independent tasks to be processed by the p slaves. All tasks are of same-size, i.e. they represent the same amount of processing. Tasks are considered to be atomic (execution cannot be preempted once initiated). Processors are heterogeneous;

Proceedings of the 2001 IEEE International Conference on Cluster Computing (CLUSTER'01)
0-7695-1116-3/02 $17.00 © 2002 IEEE

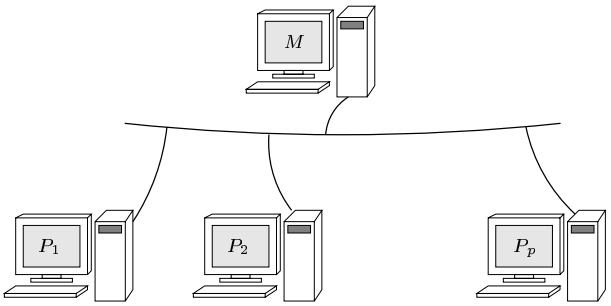

Figure 1. The target master-slave architecture.

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 smallest value of T for which the answer is at least C.

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. It is not difficult to solve both previous problems using a greedy algorithm. The solution of problem MaxTasks(T) is straightforward: we let $c_i = \left\lfloor \frac{T}{t_i} \right\rfloor$ for all i, $1 \leq i \leq p$, which obviously defines the optimal solution.

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 backpropagation neural networks on heterogeneous networks of workstations, using the PVM library [7]. 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 it 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 $0, t_{\text{com}}, 2t_{\text{com}}, \ldots, (p-1)t_{\text{com}}$. 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:

MaxTasks1(T) Given a time bound T, determine the best allocation 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:

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

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 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 $T - \sigma_2(i)t^2_{\text{com}}$. 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. 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:

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

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 such a 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.
- $\forall 1 \leq i < j,\ \ \text{if } f_{\text{proc}}(i) = f_{\text{proc}}(j) = k$, 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. 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 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:

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

- $\forall 1 \leq i < j$, 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

$$[f^1_{\text{startcomm}}(j), f^1_{\text{startcomm}}(j) + t_{\text{com}}] \cap [f_{\text{startcomp}}(i), f_{\text{startcomp}}(i) + t_k] = \emptyset \text{ and}$$
$$[f^2_{\text{startcomm}}(i), f^2_{\text{startcomm}}(i) + t_{\text{com}}] \cap [f_{\text{startcomp}}(j), f_{\text{startcomp}}(j) + t_k] = \emptyset,$$

  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. 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:

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

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 amount to sort the cycle-times as $t_1 \leq t_2 \leq \ldots \leq t_p$ and to let $\sigma(i) = i$ for $1 \leq i \leq p$. The intuition is that fastest processors execute tasks more rapidly than the others, hence 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 5 tasks, while the second processor can execute only 2.

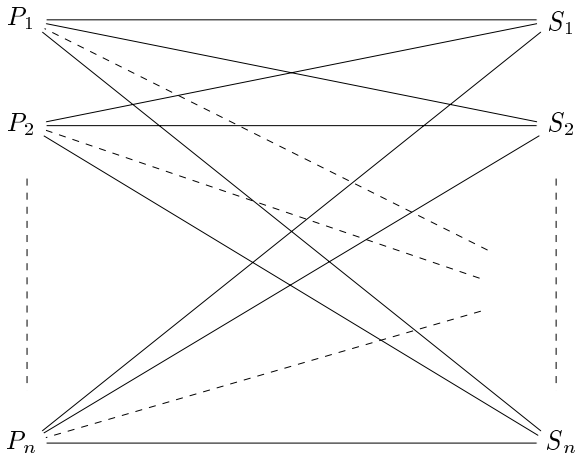

Figure 2. Bipartite graph for MaxTasks1(T)**.**

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 Figure 2. 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 $\left\lfloor \frac{T - jt_{\text{com}}}{t_i} \right\rfloor$. Extracting a matching from the graph enables to assign a different value of σ for each processor, thereby guaranteeing that σ is indeed a permutation. In fact, there is a one-to-one correspondence between matchings 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 [8, 16]. To conclude this section, we formally state our result:

Proposition 1 *The optimal solution to the* MaxTasks1(T) *problem with initial messages can be found in time of order $O(p^3)$ with p processors using the above weighted-matching algorithm.*

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 do not know of any polynomial algorithm for the general case. We present an efficient guaranteed approximation using matching techniques, as ex-

plained in Section 4.1. In Section 4.2, we give some remarks about the complexity of MaxTasks2(T).

4.1 Matching techniques

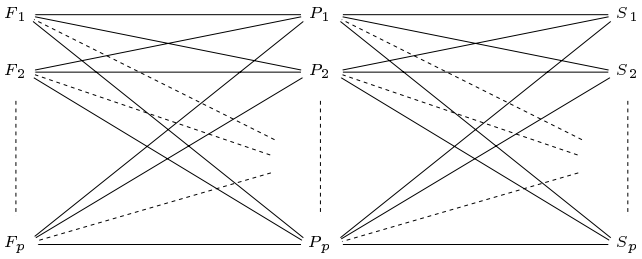

Figure 3. Bipartite graph with initial and final communications.

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 Figure 3. 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 a 2-factor from the graph [8, 16]: more precisely, we select 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 2-factor from the graph with $3p$ vertices is of order $O(p^3)$ 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 Figure 3) in time of order $O(p^3)$.

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

The proof of this theorem is detailed in [3].

4.2 Some remarks about the complexity of MaxTasks2(T)

We have not found any polynomial algorithm for the general case, and we have not been able to prove the NP-completeness of MaxTasks2(T). Nevertheless, we can formulate a few remarks about the intrinsic difficulty of MaxTasks2(T). First, an exhaustive search of all possible permutations would have a complexity of order $O((p!)^2)$, which is impossible in practice as soon as $p \geq 9$. Moreover, the problem seems to be difficult even for very simple instances of MaxTasks2(T), as shown in [3]. Indeed, let us consider the following open (polynomial vs. NP-complete) problem in combinatorial optimization (see [10]):

Permutation Sums:
Instance: Let $a_1 \leq a_2 \leq \ldots \leq a_p$ be p positive integers satisfying $\sum_{i=1}^{p} a_i = p(p+1)$.
Question: Do there exist two permutations σ_1 and σ_2 of $\{1, 2, \ldots, n\}$ such that

$$\forall i \in [1, p] : \sigma_1(i) + \sigma_2(i) = a_i.$$

In [3], it is shown that if Permutation Sums is proved to be NP-complete then MaxTasks2(T) is also NP-complete and if MaxTasks2(T) can be solved in polynomial time, then it proves that Permutation Sums can also be solved in polynomial time. Thus, we can expect that the general instance of MaxTasks2(T) is intrinsically difficult.

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

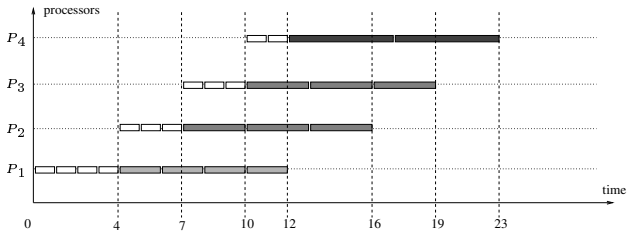

Figure 4. Example when $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} \leq 1$.

tasks that can be processed within T units of time:

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

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.

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 $\tau_i = t_{\text{com}} + t_i$, 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 τ_i: T^{pattern} determines the length of the pattern. Let $\nu_i^{\text{pattern}} = \frac{T^{\text{pattern}}}{\tau_i}$ be the number of tasks processed by processor P_i during the execution of the pattern. To formally build the pattern, we need some complicated notations. 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 $\nu_0^{\text{pattern}} = 0$):

Determine which processor executes task number i:

$$\forall 1 \leq i \leq \sum_{k=1}^{p} \nu_k^{\text{pattern}}\ :$$

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

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

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

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

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 and easily check that these functions 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 previous section. Indeed, in this case, the network is the limiting resource. The algorithm consists in 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 $t_1 \leq t_2 \leq \ldots \leq t_p$. Let τ_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 τ_i, $1 \leq i \leq p_{max}$. Moreover, define ν_i^{pattern} as follows:

$$\forall\ 1 \leq i \leq p_{max},\ \ \nu_i^{\text{pattern}} = \frac{T^{\text{pattern}}}{\tau_i},$$

$$\nu_{p_{max}+1}^{\text{pattern}} = \frac{T^{\text{pattern}} - t_{\text{com}} \sum_{i=1}^{p_{max}} \nu_i^{\text{pattern}}}{t_{\text{com}}}$$

and let

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

We see that processor number $p_{max}+1$ is not used fully, while following 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 $\left(T^{\text{pattern}} - \nu_{p_{max}+1}^{\text{pattern}} \tau_{p_{max}+1}\right)$ time steps.

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.

The following theorem states the asymptotic optimally of the algorithms defined in Section 5.2 and 5.3.

Theorem 2 *Let $N_{opt}(T)$ be the optimal number of tasks that can be executed within T time-steps. Let $N(T)$ be*

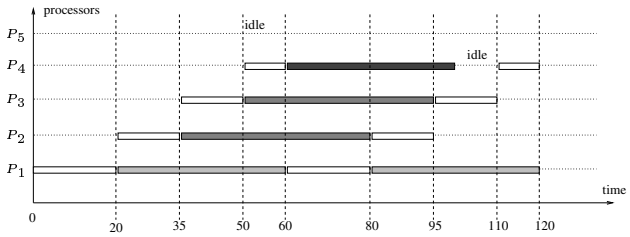

Figure 5. Example when $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} > 1$.

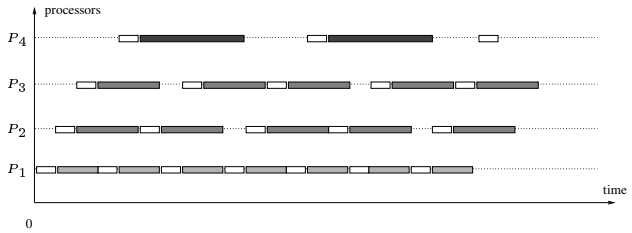

Figure 6. Execution with the greedy algorithm.

the number of tasks executed by the algorithm of Section 5.2 if $\sum_{i=1}^{p} \frac{t_{com}}{t_{com}+t_i} \leq 1$, and by the algorithm of Section 5.3 if $\sum_{i=1}^{p} \frac{t_{com}}{t_{com}+t_i} > 1$. Then

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

More details on the algorithm and on the proof of this theorem can be found in [3].

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 Section5, so we only outline the sketch of the algorithm, and describe it through an example, without detailing the proofs.

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

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

Let t^1_{com} be the communication cost for the messages from the master to the slaves, t^2_{com} 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 $t_{\text{com}} = t^1_{\text{com}} + t^2_{\text{com}}$. 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 3 *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*

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

6.1 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 Figure 6, 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 Figure 4.

The greedy algorithm also leads to a periodic allocation (the time period is 9); it is able to process 8 tasks every 9 time steps but neither the computing ressources 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 $\frac{9}{8}$.

7 Related work

To the best of our knowledge, the most related work is presented in the paper of Andonie et al [1] which we have already quoted in Section 3.

Several theoretical papers deal with complexity results for parallel machine problems with a server, establishing complexity (NP-completeness) results [9, 11, 5] and providing guaranteed approximations [12]. 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 [6]) 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. Because this instance is very specific, we do not know its complexity (polynomial versus NP-complete).

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

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. We conjecture this last problem to be intrinsically difficult 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 [2, 4]. Data decomposition, scheduling heuristics, 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. Chronopoulos, D. Grosu, and H. Galmeanu. Distributed backpropagation neural networks on a PVM heterogeneous system. In *Parallel and Distributed Computing and Systems Conference (PDCS'98)*, pages 555–560. IASTED Press, 1998.

[2] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert. Matrix-matrix multiplication on heterogeneous platforms. In *2000 International Conference on Parallel Processing (ICPP'2000)*. IEEE Computer Society Press, 2000.

[3] O. Beaumont, A. Legrand, and Y. Robert. The master-slave paradigm with heterogeneous processors. Technical Report RR-2001-13, LIP, ENS Lyon, Mar. 2001. Available at `www.ens-lyon.fr/LIP/`.

[4] P. Boulet, J. Dongarra, F. Rastello, Y. Robert, and F. Vivien. Algorithmic issues on heterogeneous computing platforms. *Parallel Processing Letters*, 9(2):197–213, 1999.

[5] P. Brucker, C. Dhaenens-Flipo, S. Knust, S. Kravchenko, and F. Werner. Complexity results for parallel machine problems with a single server. Technical Report Reihe P, No. 219, Fachbereich Mathematik Informatik, Universitt Osnabrck, 2000.

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

[7] 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*. The MIT Press, 1996.

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

[9] N. Hall, C. Potts, and C. Sriskandarajah. Parallel machine scheduling with a common server. *Discrete Applied Mathematics*, 102:223–243, 2000.

[10] S. Hedetniemi. Open Problems in Combinatorial Optimization. World Wide Web document, URL: `http://www.cs.clemson.edu/~hedet/algorithms.html`.

[11] S. Kravchenko and F. Werner. Parallel machine scheduling problems with a single server. *Mathematical Computational Modelling*, 26:1–11, 1997.

[12] H. Lee and M. Guignard. A hybrid bounding procedure for the workload allocation problem on parallel unrelated machines with setups. *Journal of the Operational Research Society*, 47:1247–1261, 1996.

[13] K. Soman, R. Fraczkiewicz, C. Mumenthaler, B. von Freyberg, and T. S. W. Braun. FANTOM - (Fast Newton - Raphson Torsion Angle Minimizer). World Wide Web document, URL: `http://www.scsb.utmb.edu/fantom/fm_home.html`. a program for "the calculation of conformations of linear and cyclic polypeptides and proteins with low conformational energies including distance and dihedral angle constraints from nuclear magnetic resonance experiments or for modeling purposes.".

[14] J. Stiles, T. Bartol, M. Salpeter, and M. Salpeter. Monte Carlo simulation of neuromuscular transmitter release using MCell, a general simulator of cellular physiological processes. *Computational Neuroscience*, pages 279–284, 1998.

[15] J. Watts and S. Taylor. A practical approach to dynamic load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 9(3):235–248, 1998.

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