HAL
open science

Master-slave Tasking on Heterogeneous Processors

Pierre-François Dutot

To cite this version:

Pierre-François Dutot. Master-slave Tasking on Heterogeneous Processors. International Parallel and Distributed Processing Symposium, Apr 2003, Nice, France. inria-00001081

HAL Id: inria-00001081

https://inria.hal.science/inria-00001081

Submitted on 1 Feb 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Master-slave Tasking on Heterogeneous Processors

Pierre-François Dutot
Laboratoire ID-IMAG
51 avenue Jean Kuntzmann
38330 Montbonnot Saint Martin, France
Pierre-Francois.Dutot@imag.fr

Abstract

In this paper, we consider the problem of scheduling independent identical tasks on heterogeneous processors where communication times and processing times are different. We assume that communication-computation overlap is possible for every processor, but only allow one send and one receive at a time. We propose an algorithm for chains of processors based on an iterative backward construction of the schedule, which is polynomial in the number of processors and in the number of tasks. The complexity is $O(np^2)$ where n is the number of tasks and p the number of processors. We prove this algorithm to be optimal with respect to the makespan. We extend this result to a special kind of tree called spider graphs.

1 Introduction

Parallel computation on heterogeneous platform is one of the most important issue in high performance computing nowadays. Famous parallel applications such as SETI@home [9] or the Mersenne prime search [8] are using a wide variety of commodity computing resources to extend as much as possible the pool of volunteers they depend on.

In this paper we deal with the problem of scheduling independent equal sized tasks, as used in the previously cited applications, on a sub-class of "grid" computing platform where communication links and computation nodes may be of any kind, and therefore have different speeds. This problem has already been adressed in [2] for the special case of fork graphs. The steady state for trees is also studied in the same paper. Some of the authors of [2] also wrote a research report [3] with a good bibliography covering many similar problems. In order to approach the more difficult problem of scheduling on general trees, we study in this paper the case of chains of processors. The results on the chains and fork graphs are then merged into an algorithm for a subset of trees called spider graphs, where only one node (the master node here) can have an arity greater than 2.

This work is also related to divisible tasks as first introduced by [5]. Robertazzi et al studied many variations around this topic for divisible tasks. In [1] they first studied the homogeneous tree problem. Then in [10] they looked at the bus problem which is identical to a fork graph with homogeneous communications and heterogeneous computations times. They also recently worked [4] on star graph with heterogeneous communications and computations times. The main difference is that we are working with quantums of workload whereas a divisible task can be divided in fractions of any size. A closer link can be made with recent results as presented in [7], where the author reduces a homogeneous grid with multi-port communication to a heterogeneous chain. Readers interested in divisible tasks may want to read [6] where an "optimal" solution used in [7] is improved.

In the next section we present the model and give some basic definitions needed for the proofs. In section 3 we describe our chain algorithm. Some basic properties of the schedule produced are given in section 4. Section 5 contains the proof of optimality of the chain algorithm. Then we recall some of the results on fork graphs in section 6 that we need for the algorithm on spider graphs which is presented and proved optimal in section 7. We briefly conclude in section 8.

2 Definitions

For now, we consider a chain of heterogeneous processors as depicted in figure 1. Each processor has an incoming link with latency c_i and needs the working time w_i to process a task. The processors are numbered from 1 to p, the first being the closest to the source of tasks.

Every processor can only have one communication on his incoming link at a time. Similarly it can only send one task at a time, and process another. In all the article $[\![i; j]\!]$ denotes the set of all integers between i and j (inclusive).

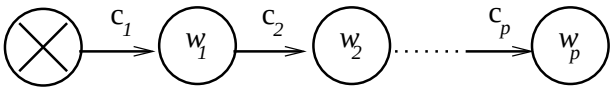

Figure 1. Chain where the first node is the master node.

Definition 1 *A schedule for n tasks on a given chain of processors of length p is a set of functions giving for every task i a processor number $P(i)$ where it will be executed, a starting time $T(i)$, and the set $C(i)$ of communication times for all the communication links taken by the task. $C(i) = \{C_1^i; C_2^i; \cdots; C_{P(i)}^i\}$ where C_j^i is the emission time of the communication from processor $j-1$ to processor j concerning task i. In the following sections $C(i)$ will be called the* communication vector

$$\begin{aligned} P(i) &: [\![1;n]\!] \rightarrow [\![1;p]\!] \\ T(i) &: [\![1;n]\!] \rightarrow \mathbb{N} \\ C(i) &: [\![1;n]\!] \rightarrow \{\mathbb{N}^i | i \in [\![1;p]\!]\} \end{aligned}$$

A schedule is *feasible* iff it satisfies the four properties:

$$\forall i \in [\![1;n]\!],\ \forall k \in [\![2;P(i)]\!] \quad C_{k-1}^i + c_{k-1} \leq C_k^i \quad (1)$$

$$\forall i \in [\![1;n]\!] \quad C_{P(i)}^i + c_{P(i)} \leq T(i) \quad (2)$$

$$\forall i,j \in [\![1;n]\!],\ i \neq j \quad P(i) = P(j) \Longrightarrow |T(i) - T(j)| \geq w_{P(i)} \quad (3)$$

$$\forall i,j \in [\![1;n]\!],\ i \neq j,\ \forall k \in [\![1;p]\!] \quad (k \leq P(i) \text{ and } k \leq P(j)) \Longrightarrow |C_k^i - C_k^j| \geq c_k \quad (4)$$

The first property states that a task cannot be reemitted by a processor before the reception is completed. The second one states that a task must have been completely received before starting the execution. The third one says that two tasks executed on a single processor cannot overlap. The last property says that the communication of two tasks on the same link cannot overlap.

Without loss of generality, we will always consider schedules where tasks are emitted from the master in their index order $C_1^1 \leq C_1^2 \leq \cdots \leq C_1^n$.

Figure 2 presents a simple schedule on a chain with two processors. The values c_i are the labels on the edges and the values w_i are the numbers on the nodes. The dashed curve denotes a delayed task, i.e. the second task has been received and buffered, and had to wait the completion of the first task before starting its execution.

The objective function is the *makespan* defined as the smallest time where all tasks are done.

Definition 2 *The makespan T_{max} is defined as follows:*

$$T_{max} = \max_{i \in [\![1;n]\!]} \left(T(i) + w_{P(i)}\right)$$

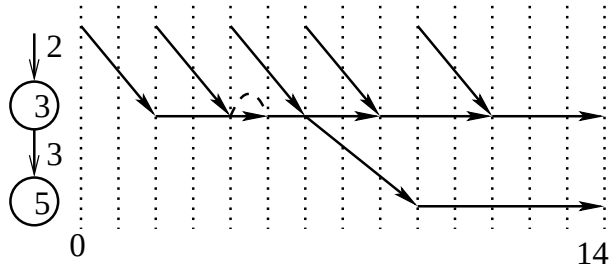

Figure 2. A representation of a schedule

Although two communication vectors may have different lengths, they can always be compared in the following way:

Definition 3 *Let $A = \{a_1, \ldots, a_i\}$ and $B = \{b_1, \ldots, b_j\}$ two different communication vectors, we say that A is inferior to B ($A \prec B$) iff one of the two following conditions is verified:*

- *$\exists k \in [\![1; min(i,j)]\!]$ such that $a_k \neq b_k$ and let l be the smallest number such that $a_l \neq b_l$, we have $a_l < b_l$*
- *$i > j$ and $\forall k \in [\![1;j]\!]\ a_k = b_k$*

3 The algorithm

Inputs : a length p, the value of the chain $(c_i)_{i \in [\![1;p]\!]}$, $(w_i)_{i \in [\![1;p]\!]}$ and the tasks to be scheduled $task_1 \ldots task_n$.

Outputs : A feasible schedule for the n tasks.

As our algorithm builds the solution from the end, we introduce a constant $T_\infty = c_1 + (n-1) \times max(w_1, c_1) + w_1$. This time is the time of the simple schedule placing all the tasks on the first processor. Our algorithm is a greedy algorithm, tasks are scheduled one after another and a scheduling decision is never reconsidered afterwards. When all the tasks are scheduled, a shift of C_1^1 units of time is applied to $T(i)$ and $C(i)$ for all i to set the starting time of the schedule to 0.

We will need two more variables for the algorithm called respectively the "hull" (h) and the "occupancy" (o). These variables are vectors of length p corresponding to the time from which the communication links (respectively the processors) might be used. They are both initialised at T_∞ for all their values.

To schedule task $task_n$ we consider the p greatest communication vectors (according to the order defined in def.3) ${}^kC(n)$ corresponding to the execution of the task on the p processors ending at time T_∞. The values of these vectors are:

$$\forall k \in [\![1;p]\!]\ {}^kC(n) = \left\{{}^kC_1^n; \cdots; {}^kC_k^n\right\}$$

with

$$\forall i \in [\![1;k]\!]\ {}^kC_i^n = T_\infty - w_k - \sum_{j=i}^{k} c_j$$

Among these p communication vectors we take the greatest (there is only one as their length differ) and set $C(n)$ with it.

```
   T∞ = c1 + (n-1) * max(w1,c1) + w1

// Initialisation of h and o vectors.
   for i = 1 to p do
      hi = oi = T∞
   endfor

// Initialisation of C(i)
   for i = 1 to n do
      C(i) = {0;···;0}
   endfor

// Computation of the
// communication vectors
   for i = n downto 1 do
      for k = p downto 1 do
         ^kC^i_k = min(o_k − w_k − c_k, h_k − c_k)
         for j = k-1 downto 1 do
            ^kC^i_j = min(^kC^i_{j+1} − c_j, h_j − c_j)
         endfor
         if C(i) ≺ ^kC(i)
            then C(i) = ^kC(i)
      endfor
      P(i) = length(C(i))
      T(i) = o_{P(i)} − w_{P(i)}
      o_{P(i)} = T(i)
      for k = 1 to P(i)
         hk = C^i_k
      endfor
   endfor

// Apply the time shift
   for i = n downto 1 do
      T(i) = T(i) − C^1_1
      for k = P(i) downto 1 do
         C^i_k = C^i_k − C^1_1
      endfor
   endfor

return C,P and T
```

Figure 3. The algorithm in pseudo-code.

The task is placed on the processor $P(n)$ corresponding to this communication vector $C(n)$ with time $T(n) = T_\infty - w_{P(n)}$. The hull and occupancy are updated with:

$$o_{P(n)} = T(n) \quad \forall i \in [\![1;P(n)]\!]\ h_i = C_i^n$$

When $task_j$ has been scheduled, we place $task_{j-1}$ in the following way. First we compute the ${}^kC(j-1)$ vectors with:

$${}^kC_k^{j-1} = min\left(o_k - w_k - c_k, h_k - c_k\right)$$

$${}^kC_i^{j-1} = min\left({}^kC_{i+1}^{j-1} - c_i, h_i - c_i\right)$$

Then we choose the greatest ${}^kC(j-1)$ and set $C(j-1)$ and $P(j-1)$ accordingly. Finally we set $T(j-1) = o_{P(j-1)} - w_{P(j-1)}$.

The vectors o and h are updated with:

$$o_{P(j-1)} = T(j-1) \quad \forall i \in [\![1;P(j-1)]\!]\ h_i = C_i^{j-1}$$

To prove that the schedule is feasible, we just have to show that it verifies the four conditions given earlier. This proof is left to the reader.

The algorithm is written in pseudo-code in the figure 3. The complexity of this algorithm is $O(np^2)$, as the vector comparison is in $O(p)$.

4 Properties of the schedule

Before going into the detailled proof of the optimality of our algorithm, we need to describe some useful properties of the schedule our algorithm outputs.

Lemma 1 *Let h be a communication hull, k and l two processors, and i a task. If ${}^kC(i) \prec {}^lC(i)$, then for every $q \leq min(k,l)$ we can prove $\{{}^kC_q^i;\cdots;{}^kC_k^i\} \prec \{{}^lC_q^i;\cdots;{}^lC_l^i\}$.*

The idea motivating this lemma is illustrated in figure 4. There should be no crossing in two possible communication vectors of the same task, as for all k the communication vector ${}^kC(i)$ is the greatest of all possible vectors. With the

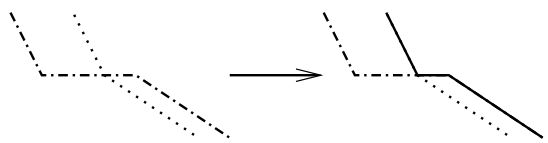

Figure 4. There is always a better solution than a crossing

two vectors drawn on the figure, we can see that a better

option for the dash-dotted vector would be the vector drawn with a solid line.

As k and l are two different processors, the length of the communication vectors differs. If ${}^{l}C(i)$ is a prefix of the ${}^{k}C(i)$ the lemma is verified.

Here ${}^{k}C(i)$ cannot be a prefix of ${}^{l}C(i)$ because ${}^{k}C(i) \prec {}^{l}C(i)$. Therefore if ${}^{l}C(i)$ is not a prefix then there is an r such that ${}^{k}C_r^i < {}^{l}C_r^i$ and all smaller terms are equal. As ${}^{\cdot}C_r^i$ is defined in the algorithm as $min({}^{\cdot}C_{r+1}^i - c_r, h_r - c_r)$, ${}^{k}C_r^i < {}^{l}C_r^i$ implies ${}^{k}C_{r+1}^i < {}^{l}C_{r+1}^i$. This inequality can be proved recursively for all terms until the end of the smallest vector, which concludes the proof of the lemma. □

Lemma 2 *Let $(c_i)_{i\in[\![1;p]\!]}$, $(w_i)_{i\in[\![1;p]\!]}$ be a chain of processors, and n tasks scheduled by our algorithm on this chain. Let n' be the number of task of this schedule verifying $P(i) \geq 2$. The schedule of these tasks is the same as the schedule of n' tasks on the sub-chain $(c_i)_{i\in[\![2;p]\!]}$, $(w_i)_{i\in[\![2;p]\!]}$ with our algorithm, with a time shift of $T_{shift} = \min_{(i/P(i)\geq 2)}(C_2^i)$.*

More formally, let $(\sigma(i))_{i\in[\![1;n']\!]}$ be the n' tasks of the sub-chain, putting a hat on the variables concerning the sub-chain we have:

$$\begin{aligned} \forall i \in [\![1;n']\!] \quad & \hat{P}(i) = P(\sigma(i)) \\ \forall i \in [\![1;n']\!] \quad & \hat{T}(i) = T(\sigma(i)) - T_{shift} \\ \forall i \in [\![1;n']\!]\ \forall q \in [\![2;\hat{P}(i)]\!] \quad & \hat{C}_q^i = C_q^{\sigma(i)} - T_{shift} \end{aligned}$$

The proof of this lemma derives from the previous lemma. Looking at the algorithm, we can see that ${}^{k}C_2^i$ does not depend on the value of ${}^{k}C_1^i$, as we compute the ${}^{k}C_q^i$ with q going from k downto 1. The execution of the algorithm on the chain $(c_i)_{i\in[\![1;p]\!]}$, $(w_i)_{i\in[\![1;p]\!]}$ is the same as the execution of the algorithm on $(c_i)_{i\in[\![2;p]\!]}$, $(w_i)_{i\in[\![2;p]\!]}$ (as with the previous lemma we proved that if ${}^{k}C(i) \prec {}^{l}C(i)$ we have $\{{}^{k}C_2^i; \cdots; {}^{k}C_k^i\} \prec \{{}^{l}C_2^i; \cdots; {}^{l}C_l^i\}$), with the exception of the $n - n'$ tasks which will be placed on the first processor. The biggest difference is the first time reference and the final time shift which give a time lag between the two schedules. □

5 Proof of optimality

Theorem 1 *The heuristic given in section 3 is optimal with respect to the makespan (termination date of the last task).*

For the limit cases when $p = 1$ or $n = 1$ the algorithm is clearly optimal. In the first case the processor is filled either without delays in the communications or without delays in the computation. In the second case the algorithm picks one of the processors on which the makespan is minimal.

The proof of the general case is based on the assumption that there is a chain and a number n such that the optimal schedule is faster than our schedule. We will show that this assumption brings a contradiction.

Among all the chains for which our algorithm is not optimal we choose one where p is minimal. As we said above $p > 1$, therefore our algorithm is optimal for the non-empty set of chains of length strictly less than p.

For the considered chain, let n be the smallest number of tasks for which our algorithm is not optimal. Again, we have $n > 1$ as our algorithm is optimal for a single task.

Let $T_{max}^{opt(n)}$ be the optimal makespan for n tasks and $T_{max}^{alg(n)}$ be the makespan of our algorithm. We supposed that $T_{max}^{opt(n)} < T_{max}^{alg(n)}$. As we considered the smallest n, we have $T_{max}^{opt(n-1)} = T_{max}^{alg(n-1)}$.

At this point we need to introduce another notation. We add a lower left indice when needed on all the previous notations when a value is corresponding to a given heuristic. For example ${}_{opt(n)}P(i)$ is the processor on which the task i is scheduled in the optimal schedule for n tasks and ${}_{alg(n-1)}^{k}C_q^i$ is the emission time of the communication concerning task i between processors $q-1$ and q if the task is to be placed on processor k with our algorithm on $n-1$ tasks.

When we remove the first task of the schedule $opt(n)$ we have a schedule of $n-1$ tasks, which is necessarily longer or equal to the schedule of our algorithm on the $n-1$ last tasks, as we said that our algorithm was optimal for $n-1$ tasks. This is written: $T_{max}^{opt(n)} - {}_{opt(n)}C_1^2 \geq T_{max}^{alg(n-1)} = T_{max}^{alg(n)} - {}_{alg(n)}C_1^2$.

Equation (4) from definition 1 implies that ${}_{opt(n)}C_1^2 \geq {}_{opt(n)}C_1^1 + c_1 \geq c_1$. Which means that the full communication of the first task has to be completed before the second task can be emitted. Rewritting this with the previous equation we have: ${}_{alg(n)}C_1^2 \geq T_{max}^{alg(n)} - T_{max}^{opt(n)} + c_1$.

So if our algorithm does not give an optimal schedule, there is some idle time on the first communication link between the emissions of the first and the second task. We will now consider the two cases $P(1) = 1$ and $P(1) \geq 2$.

- If $P(1) = 1$ the first processor has no idle time in computation, or else (as the task are executed as late as possible) the communication would have been delayed. This implies that we did one more task on the first processor than the optimal schedule (or else the total time would be equal). Thus the optimal algorithm did one more task than we did on the sub-chain $(c_i)_{i\in[\![2;p]\!]}$, $(w_i)_{i\in[\![2;p]\!]}$ within a total time less or equal to $T_{max}^{alg(n)} - c_1 - 1$. As our algorithm could not do as many tasks within the same time limits (or else we

would not have $P(1) = 1$ as one ${}^kC_1^1$ would be greater than ${}^1C_1^1$) it is not optimal for a number of tasks less or equal to n on the sub-chain $(c_i)_{i\in[\![2;p]\!]}$, $(w_i)_{i\in[\![2;p]\!]}$, which contradicts the hypothesis that p is minimal.

- If $P(1) \geq 2$ it implies that the task could not be placed on the first processor because it is fully loaded (the remaining avalaible time is less than w_1) and that our algorithm was less efficient than the optimal on the sub-chain with a number of task lesser or equal to the number placed by the optimal on this sub-chain (we loaded the first processor as much as possible, but don't know if the optimal schedule did). Which again contradicts the hypothesis that p is minimal.

The two cases lead to the conclusion that the hypothesis was wrong, and this ends the proof of theorem 1. □

6 Extension to spiders

A spider is a special kind of tree where only the master node is allowed to have several children. A very simple example is presented in figure 5. In this article, the master node will always be the root of the spider. A feasible schedule on a spider graph is a schedule where each node only sends one task at a time and each processor computes one task at a time as before. The added difficulty here is that the master node is connected to many children and must choose for each task the child that will receive it.

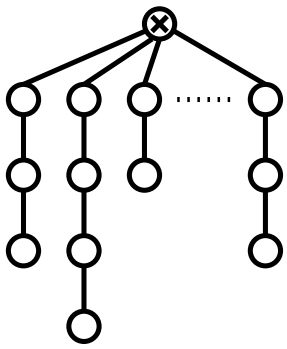

Figure 5. A spider.

Mixing the previous algorithm with another one already published by another research group [2] concerning fork graphs, we can achieve an optimal algorithm for the scheduling problem on spiders which is polynomial in the number of tasks and the number of processors.

We will first briefly recall how the algorithm for fork-graphs was designed before going into the details of our algorithm on spiders. First, it takes as input a fork graph, a number n of tasks and a time limit T_{lim}. The algorithm succeeds if it can schedule n tasks with a makespan lower or equal to T_{lim}. If it cannot then no such schedule is possible.

The algorithm on fork-graphs is based on a tranformation of the problem where any processor can compute any number of tasks into a problem where there are more processors which can compute only one task each. This transformation is easily done replacing every processor (c_i, w_i) with a set of processors with the same value for the incoming link and different processing times as represented on figure 6 (where $m_i = max(c_i, w_i)$).

With a transformed instance, it is clear that the communication time is the only resource shared by all tasks. Therefore the communication usage should be minimized. Another interesting property is that any feasible schedule can be transformed into another feasible schedule where the tasks are sorted in decreasing order of processing times of the different processors they are alloted to.

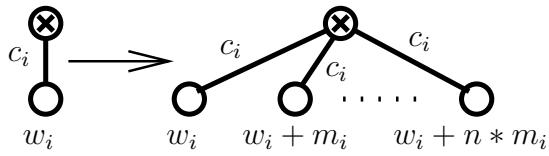

Figure 6. Transformation of a single node.

Built on those two properties the algorithm can be described in the following terms. Given a time bound T_{lim}, sort the processors by ascending communication times and break ties by sorting according to processing times. Then allocate tasks to processors in this order whenever possible until no more task can be added. To know if it is possible to add a task on a given processor, you have to check if the insertion of the communication time in the schedule is possible when tasks are ordered by processing times.

The interested reader should refer to the original paper [2] for the complexity analysis, optimality proof and formal presentation of this algorithm.

7 Spider algorithm

As the fork graph algorithm, our spider algorithm will take as additionnal input a time limit T_{lim}.

The first thing to do before merging both algorithms in one is to rewrite our chain algorithm to take a time limit as input as well as a number of tasks n and output the schedule with the biggest number of tasks possible within the time limit, or the schedule for n tasks if it is feasible in at most T_{lim} time units. To achieve this, we have to change T_∞ to that time limit $T_{lim}$and change the first for in the computation of communication vectors to a $while$ and stop when a task gets a negative emission time C_1^i or when n tasks have been scheduled.

Let us now consider what has to be done to design an algorithm for spiders. The algorithm on fork graphs was based on a transformation of the original fork. This decomposition induces many single-task nodes with different execution times. After a first run of our algorithm on chains, we can make a similar transformation and see a chain as a fork

graph with all communication link set to c_1 and processing times equals to $T_{lim} - C_1^i - c_1$. This transformation gives us a fork where as many tasks can be scheduled in the same time interval as on the chain.

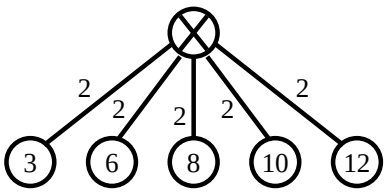

Figure 7. Transformation of the example of figure 2.

This transformation for the example given in figure 2 is depicted in figure 7. For example the task that was scheduled on the second processor corresponds to the node with processing time 8.

All the chains leaving from a single master node in the spider graph can be transformed in this way to form a fork graph with single task nodes with communication times depending on the chain there are issued from. The maximum time T_{lim} being defined, we compute for each chain an optimal schedule for T_{lim} time units, and then we create for each scheduled task a single task node with a communication link and processing time as defined above.

Then we can apply the fork algorithm to choose the nodes where an execution shall take place and reverting to the original spider we can relate those nodes to an actual schedule.

More formally the algorithm can be written:

```
(1) Given T_lim, n and a spider
(2) For each chain of the spider
        compute n, C, P, and T
(3) Create the associated fork graph
(4) Compute the optimal schedule on
        the fork graph
(5) Revert to a spider schedule
```

Theorem 2 *The algorithm on spider graph is polynomial in the number of tasks and the number of processors.*

The complexity of line 2 is $\sum_c np_c^2$, where p_c is the length of the chain c. $\sum_c np_c^2 < np^2$ with p the total number of nodes in the spider. Line 3 is a simple rewriting of the results in the previous line. Line 4 is quadratic in the number of single task slaves, which is here bounded by kn, where k is the arity of the master node. So the complexity of line 4 is bounded by $n^2k^2 < n^2p^2$. Line 5 is again a simple rewriting of the results in the previous line and is of lower complexity. The overall complexity is lower than $O(n^2p^2)$. □

Lemma 3 *A feasible schedule for the fork graph gives a feasible schedule for the spider graph.*

To each node of the fork graph is associated the allocation of the corresponding task in one of the chains. For a given chain, we can look at all the corresponding nodes taken in the fork schedule. The resulting schedule on the chain is very similar to the optimal schedule given by our algorithm, from which some tasks have been removed. The only modification is that the emission times on the first link are choosen by the fork graph algorithm. But as we decided that the processing time for the node associated to task i on the chain we are considering is $T_{lim} - C_1^i - c_1$, the emission time choosen by the fork graph is necessarily less or equal to C_1^i. As the optimal schedule was feasible, the same with some tasks removed and the first communications possibly done earlier (but without any conflict on the link as ensured by the fork algorithm) is still feasible. This proves that there are no tasks overlapping in computation or in communication within the chain. All we need to check is that the master node complies with the "one communication at a time" policy as it sends tasks to possibly many chains. This is ensured by the fork algorithm since we do not change the emission times of the tasks on the first communication link. □

Lemma 4 *A feasible schedule on the spider can be transformed into a feasible schedule on the fork graph.*

This lemma can be a little surprising, because a schedule on the spider can have any kind of structure unrelated to the schedules built by our algorithm on chains, whereas the fork graph is built according to these schedules on chains. Yet another useful property of our algorithm on chains is that an optimal solution for n tasks is iteratively built on the optimal solutions for $n-1$ to 1 tasks. This is because we are constructing the solutions from the end of the schedule going backward in time. Getting back to the proof, let us consider one of the chains of the spider. The latest task scheduled on this chain is scheduled in at least $T_{lim} - C_1^n$ units of time (where C_1^n is the emission time of the last task scheduled by the chain algorithm) as our algorithm on chains is optimal for one task. For each task scheduled on the chain we can prove in the same way that if there are $n-i$ tasks scheduled after it then the task is scheduled in at least $T_{lim} - C_1^i$ units of time. Therefore any task scheduled on this chain can be associated with one of the single task node and still complete its execution before the time limit T_{lim}.

As this is true for all the chains of the spider, this concludes the proof of the lemma. □

Theorem 3 *The schedule produced by our algorithm on spider graph is optimal.*

The proof to this theorem is a simple conclusion of previous steps. As shown in lemma 4 any schedule on the spider can be related to a schedule on the associated fork graph. This holds for the optimal schedule on the spider within T_{lim} units of time. The schedule on the fork associated with the optimal on the spider is not necessarily optimal for the fork. Therefore as we compute the optimal schedule on the fork graph we are doing at least as many tasks on the fork as the spider's optimal. And reverting to the spider schedule as shown in lemma 3 we are producing a feasible schedule on the spider which has at least as many tasks as the spider's optimal. □

8 Conclusion

In this article we provided an algorithm polynomial in the number of processors and in the number of tasks to give the optimal schedule for identical independent tasks on a chain of heterogeneous processors. Building on previous work from other authors we extended this algorithm to a special kind of trees called spider graphs. This is a first step toward solving the more general problem of trees of processors.

The long term objective of this work is to provide good heuristics for scheduling on complicated graphs of heterogeneous processors, by covering those graphs with simpler structures.

References

[1] 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 Transactions on computers*, 43(10):1184–1196, October 1994.

[2] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Bandwidth-centric allocation of independent tasks on heterogeneous platforms. In *International Parallel and Distributed Processing Symposium*, 2002. Technical report avalaible at `http://www.ens-lyon.fr/~yrobert`.

[3] O. Beaumont, A. Legrand, and Y. Robert. Static scheduling strategies for heterogeneous systems. Technical Report 2002-29, École Normale Supérieure de Lyon, July 2002. Avalaible at `http://www.ens-lyon.fr/~yrobert`.

[4] S. Charcranoon, T. G. Robertazzi, and S. Luryi. Optimizing computing costs using divisible load analysis. *IEEE Transactions on computers*, 49(9):987–991, September 2000.

[5] Y. Cheng and T. Robertazzi. Distributed computation for a tree network with communication delays. *IEEE Trans. on Aerospace and Electronic Systems*, 24(6):700–712, 1988.

[6] P. Dutot. Divisible load on heterogeneous linear arrays corrected. Technical report, Laboratoire Informatique et Distribution, 2002. URL: `http://www-id.imag.fr/Laboratoire/Membres/Dutot_Pierre-Francois`.

[7] K. Li. Scheduling divisible tasks on heterogeneous linear arrays with applications to layered networks. In *Workshop on Parallel and Distributed Scientifi c and Engineering Computing with Applications*, 2002.

[8] Mersenne Prime Search. URL: `http://www.mersenne.org`.

[9] SETI at home. URL: `http://setiathome.ssl.berkeley.edu`.

[10] J. Sohn, T. G. Robertazzi, and S. Luryi. Optimizing computing costs using divisible load analysis. *IEEE Transactions on parallel and distributed systems*, 9(3):225–234, March 1998.