Scheduling trees with large communication delays on two identical processors *

F. Afrati [1] E. Bampis [2] L. Finta [3] I. Milis [4]

[1] National Technical University of Athens, 9, Heroon Polytechniou str., 15773, Athens, Greece
[2] LaMI, Université d'Evry, Boulevard des Coquibus, 91025 Evry Cedex, France
[3] LIPN, Université Paris 13, Avenue Jean-Baptiste Clément, 93430 Villetaneuse Cedex, France
[4] Athens University of Economics and Business, 76, Patission str., 10434 Athens, Greece

Abstract. We consider the problem of scheduling trees on two identical processors in order to minimize the *makespan*. We assume that tasks have unit execution times, and arcs are associated with large identical integer communication delays. We prove that the problem is NP-hard in the strong sense even when restricted to the class of *binary trees*, and we provide a polynomial-time algorithm for *complete* binary trees.

1 Introduction

Two-processor scheduling is one of the most known problems in scheduling theory. It is a particular case of the famous m-processor scheduling problem where a task graph has to be scheduled on m identical processors in such a way that the *makespan* is minimized. Tasks are considered uniform, having unit execution times (UET).

In the case where the interprocessor communication delays are not taken into account, i.e. when the arcs of the task graph are associated with zero communication delays, the two-processor scheduling problem is polynomial [8] [4]. On the contrary, the complexity of the three-processor scheduling problem remains an outstanding open question [10].

This picture changes when we consider the two-processor scheduling problem with unit interprocessor communication delays. This variant of the problem is extensively studied [3] [20] [17] [13] [14]. However, its complexity for arbitrary task graphs remains unknown and polynomial time optimal algorithms have been shown for several classes of task graphs, especially trees [18] [12] [16] [19] [13], interval orders [16] [1] and series-parallel graphs (SP1) [7].

Although a large amount of work is concentrated on the unit communication delays case, no results are known for the two-processor scheduling problem with large communication delays. The only known results on scheduling in the presence of large communication delays concern the case where a sufficiently large number of processors is available [2] [5] [6] [15] [9].

*This work has been partially supported by the French-Greek bilateral exchange program PLATON and the GDR-PRS "Ordonnancement pour le parallélisme" program of the French government.

In this paper we deal with the two-processor scheduling problem in the presence of large integer identical communication delays. We prove that the problem becomes strongly NP-hard even for binary trees, and we present a polynomial algorithm for complete binary trees. Our results show that the complexity behavior of the two-processor scheduling problem with large interprocessor communication delays is analogous to the case where a sufficiently large number of processors is available [6].
The paper is organised as follows: we give below the formal definition of the considered scheduling model. In Section 2, we present our NP-hardness result, and in Section 3, we propose the polynomial-time algorithm.

1.1 Problem definition and notations

In this paper we assume the following model incorporating communication delays first introduced by Papadimitriou and Yannakakis [15]: We are given a set $V = \{1, 2, ..., n\}$ of partially ordered tasks and two identical processors M_1 and M_2. The set of tasks is usually described by a directed acyclic graph (dag) $\mathcal{G} = (V, E)$, referred as *task graph*. Tasks have unit execution times (UET), denoted by $p_j = 1$, and their execution is subject to precedence constraints and communication delays; whenever two communicating tasks i, j, with $(i, j) \in E$, are scheduled on different processors an integer identical communication delay $c_{ij} = c$ is introduced.
A schedule is defined as a function $\sigma : V \to N \times \{M_1, M_2\}$, i.e. $\sigma(i) = (t_i, M_i)$ where t_i is the time at which task $i \in V$ starts its execution and $M_i \in \{M_1, M_2\}$ is the processor on which task i is assigned. A schedule is a *feasible* one if:
(i) $(t_i, M_i) \neq (t_j, M_j)$, for all $i, j \in V$, $i \neq j$, and
(ii) $t_i + 1 + c \times \mathbf{1}(M_i \neq M_j) \leq t_j$, for all $(i, j) \in E$,
where $\mathbf{1}(.)$ is the indicator function.
The reverse function $\sigma^{-1}(t, M_i), i = 1, 2$, gives the task that is executed on processor M_i at the time slot $[t, t + 1[$. If no task is executed on M_i during this time unit we consider processor M_i executing a fictitious task, labeled 0. In this case processor M_i is said to be *idle* in time slot $[t, t+1[$ and such a time slot of processor M_i is called *idle time slot*.
For a feasible schedule σ by C_i, $i = 1, 2$, we denote the time by which all tasks scheduled on processor M_i finish their execution, i.e. $C_i = \max\ \{t + 1 \mid \sigma^{-1}(t, M_i) \neq 0\}, i = 1, 2$. By C_{max} we denote the length (makespan) of σ, i.e. $C_{max} = \max\{C_1, C_2\}$, and by OPT the length of an optimal schedule, i.e. OPT=$\min_\sigma\{C_{max}\}$.
According to the three-field notation scheme for scheduling problems, introduced in [11] and extended in [21], our problem is denoted as $P2 \mid prec,\ p_j = 1,\ c_{ij} = c \mid C_{\max}$.

2 NP-hardness result

In this section we prove that even the special case of our problem where the underlying task graph is restricted to a binary tree, i.e. $P2 \mid binary\ trees,\ p_j = 1,\ c_{ij} = c \mid C_{max}$, is strongly NP-hard.

In fact, we give a reduction from the following special case of the EXACT-COVER BY 3-SETS (X3C) problem, which we call X3C1 and it also is known to be NP-complete [10]:
INSTANCE: A set $U = \{1, 2, \ldots, 3m\}$ and a family $F = \{S_1, S_2, \ldots, S_k\}$, $k \geq m$, of subsets of U such that $S_t = \{i, j, v\}$, $1 \leq t \leq k$, $i, j, v \in U$, and every element of U belongs to no more than three elements of F.
QUESTION: Are there m elements of F whose union is exactly U?
For every instance of X3C1, we construct an instance of $P2 \mid \textit{binary trees},\ p_j = 1,\ c_{ij} = c \mid C_{max}$ in the following way:
We choose constant integers $G = m^{15}$, $a = m^{10}$ and $G' = m^4$ such that $G >> a >> G' >> m^3$.
For every element $S_t = \{i, j, v\}$ of F (w.l.o.g. we assume that $i > j > v$), we construct a subtree T_t as shown in Figure 1, where $H_\ell = a(m^3 + \ell)$, $\ell \in \{i, j, v\}$ (the lengths of these chains are depicted in the figure).

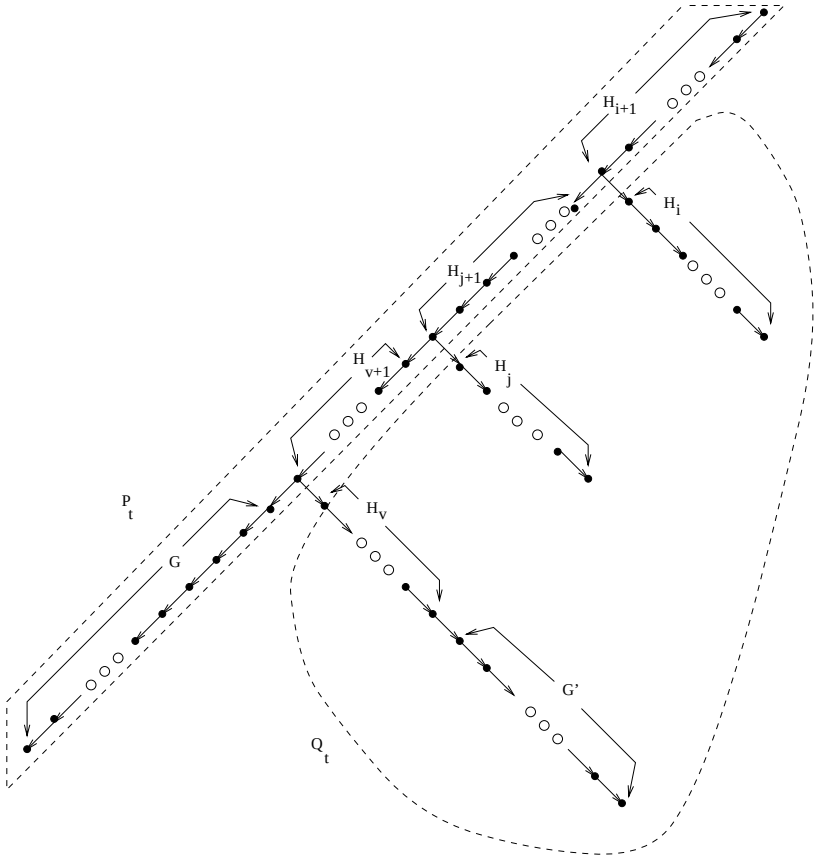

Figure 1: *The subtree T_t corresponding to the element $S_t = \{i, j, v\}$ of F*

Let us denote by $\mathcal{B}$ the total number of tasks in all the trees T_t, $t = 1, \ldots, k$, i.e.

$$\mathcal{B} = \sum_{t=1}^{k} |T_t| = kG + kG' + \sum_{\forall S_t = \{i,j,v\} \in F} (H_i + H_j + H_v + H_{i+1} + H_{j+1} + H_{v+1}).$$

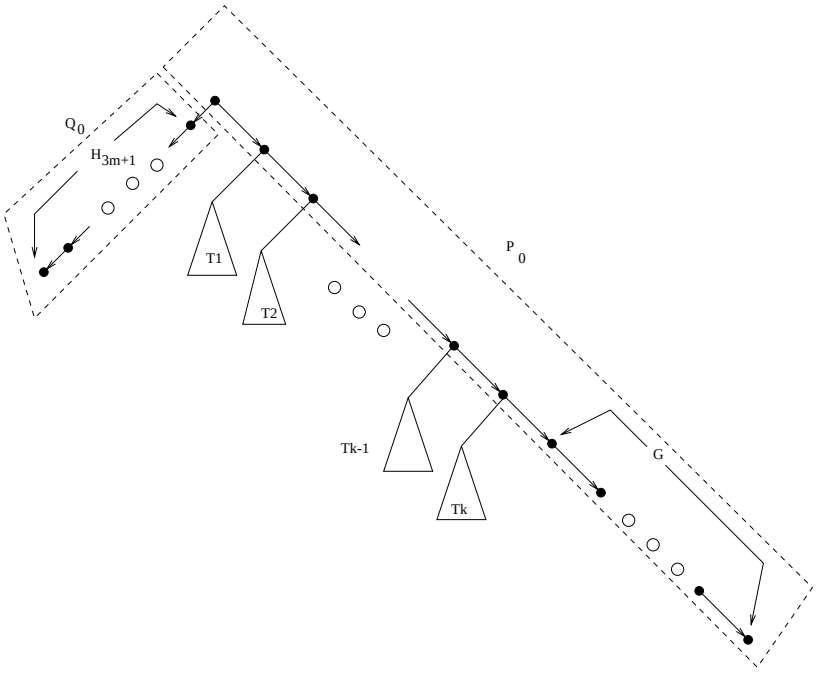

Figure 2: *The tree $\mathcal{T}$ corresponding to the instance of X3C*

Given the subtrees T_t, $t = 1, \ldots, k$, we construct the tree $\mathcal{T}$ as shown in Figure 2 and we consider the following scheduling problem:
Can we schedule $\mathcal{T}$ with communication delay

$$c = \mathcal{B} + G - 2(\textstyle\sum_{i=1}^{3m} H_i + mG') - H_{3m+1},$$

on two identical processors M_1 and M_2 within deadline

$$D_m = k + 1 + \mathcal{B} - (\textstyle\sum_{i=1}^{3m} H_i + mG') + G?$$

We prove first that if X3C1 has a positive answer, then there exists a feasible schedule $\mathcal{S}$ such that $C_{\max}(\mathcal{S}) \leq D_m$. Indeed, let, w.l.o.g., $F^* = \{S_1, S_2, \ldots, S_m\}$ be a solution of X3C1, i.e. $|F^*| = m$, $F^* \subset F$ and $\bigcup_{i=1}^{m} S_i = U$. Let us also denote by $T_1, \ldots, T_m$ the subtrees of $\mathcal{T}$ corresponding to the elements of F^* and by $P_1, \ldots, P_m$ their paths P_t, $1 \leq t \leq m$. Then, a valid schedule is the following:
Processor M_1, starts at time 0 by executing the root of the tree. Then it executes the first k tasks of the path P_0 of $\mathcal{T}$ and then the chains $H_{i+1}, H_{j+1}, H_{v+1}$ of all P_t's, $1 \leq t \leq m$, in decreasing order of their lengths i.e. $H_{3m+1}, H_{3m}, \ldots, H_2$. These $3m$ chains can be executed in that way since by the construction of $\mathcal{T}$, after the execution of some chain, say H_{l+1} of some P_t, $1 \leq t \leq m$, the chain H_l of some other $P_{t'}$, $1 \leq t' \leq m$, is always available. Processor M_1 finishes by executing the remaining G tasks of every path P_t, $t = 1, \ldots, m$, the tasks of the remaining subtrees T_t, of $\mathcal{T}$, $t = m + 1, \ldots, k$, and the last G tasks of P_0.

Processor M_2 starts executing at time $c+k+1$ the H_{3m+1} tasks of Q_0 and then the chains H_i, H_j, H_v of all Q_t's, $1 \leq t \leq m$, in decreasing order of their lengths i.e. $H_{3m}, H_{3m-1}, \ldots, H_1$. A chain of a

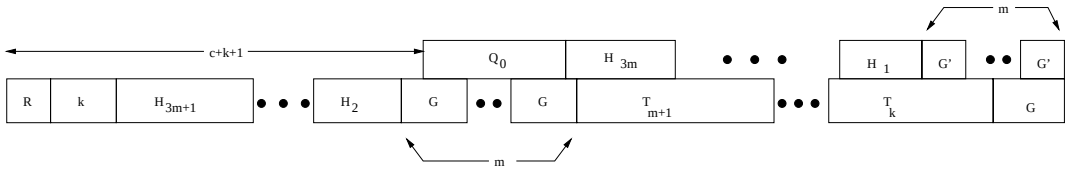

Figure 3: *A feasible schedule (R represents the root of $\mathcal{T}$, and k the first k tasks –after the root– of P_0)*

Q_t, $1 \leq t \leq m$, of length H_l is always just in time to be executed on M_2, i.e. the first task of this chain is ready to be executed exactly at the end of the execution of the last task of a chain of some $Q_{t'}$, $1 \leq t \leq m$, of length H_{l+1} on the same processor. Notice that the last task of the chain of length H_{l+1} of P_t has been completed exactly c time units before on M_1. Finally, it executes the remaining G' tasks of every Q_t, $1 \leq t \leq m$ (the order is not important). Figure 3 illustrates the Gantt-chart of such a schedule.

Let us now show that if $\mathcal{T}$ can be scheduled within time D_m, then X3C1 has a solution F^*. In the following, we consider w.l.o.g. that the root of $\mathcal{T}$ is assigned to processor M_1. We proceed by proving a series of claims concerning such a schedule:

Claim 1: Communications can be afforded only from processor M_1 to processor M_2.
Proof. Every schedule that contains communication transfers from both M_1 to M_2 and vice versa has a length of at least $2c+3$ time units. With respect to the chosen values of c and D_m, it follows that $2c + 3 > D_m$, a contradiction to the assumption on the schedule length. ∎

Claim 2: Processor M_2 may remain idle for at most k time slots after time $c + 1$.
Proof. If this is not the case, it follows that either processor M_1 has to execute more than D_m tasks or M_2 executes some tasks after time D_m. Thus, the deadline D_m is not satisfied, a contradiction. ∎

Claim 3: All the tasks of every path P_t, $0 \leq t \leq k$ are executed by processor M_1. Therefore, M_2 executes only tasks of Q_t's, $0 \leq t \leq k$.
Proof. By Claim 1, only one communication is allowed in any path from the root of $\mathcal{T}$ to a leaf. The root of $\mathcal{T}$ is executed on M_1. Thus, if a task of some P_t, is executed on M_2, then all successors of this task must be executed on M_2 as well. In this case case and since every path P_t ends up with a chain of length G, the length of the schedule will be at least $c + G > D_m$. This stands again in contradiction to the assumption on the schedule length. ∎

Claim 4: Processor M_2 executes at least $(H_{3m+1} - k)$ tasks of Q_0.
Proof. Processor M_2, by Claim 3, executes only tasks of Q_t's, $0 \leq t \leq k$, and, by Claim 2, must start not later than time $c + 1 + k$. It is clear that M_2 cannot start by this time executing tasks of a chain H_i of some Q_t, $t \neq 0$, since in such a case a chain H_{i+1} of the corresponding path P_t, have to be completed on M_1 by time $k + 1$, a contradiction since $H_{i+1} = a(m^3 + i + 1) >> m^6 >> k$. Therefore, processor M_2 will start by executing tasks of Q_0, otherwise it will remain idle for more

than $c+1+k$ time units. Processor M_1 can also execute at most k tasks of Q_0 by time $k+1$. Since, by Claim 1, communications from M_2 to M_1 are not allowed, processor M_2 will execute all the rest tasks of Q_0, that is at least $(H_{3m+1}-k)$ tasks of Q_0. ∎

Claim 5: Processor M_2 executes exactly $3m$ chains $H_s, 1 \leq s \leq 3m$, of exactly m Q_t's, $1 \leq t \leq k$.
Proof. Let E be the number of tasks executed on processor M_2. By Claim 2, M_2 executes at most $D_m-(c+1) = \sum_{i=1}^{3m+1} H_i + mG' + k$ tasks and at least $D_m-(c+1+k) = \sum_{i=1}^{3m+1} H_i + mG'$ tasks, i.e. $\sum_{i=1}^{3m+1} H_i + mG' \leq E \leq \sum_{i=1}^{3m+1} H_i + mG' + k$.
By Claim 3, processor M_2 executes only tasks of Q_t's, $0 \leq t \leq k$, that is tasks of chains $H_s, 1 \leq s \leq 3m+1$ and chains G'. Therefore, processor M_2 can execute exactly $3m+1$ chains $H_s, 1 \leq s \leq 3m+1$ and m chains G' of Q_t's, $0 \leq t \leq k$, for otherwise the bounds on E above will be violated, since $mG' << H_s, 1 \leq s \leq 3m+1$. Taking into account Claim 4 concerning Q_0, the first part of the claim follows.
For the second part of the claim observe that processor M_2 can execute a chain G' of some Q_t, $1 \leq t \leq k$, only if it has already executed the other three chains of Q_t. Since M_2 executes exactly $3m$ chains $H_s, 1 \leq s \leq 3m$ and m chains G' of Q_t's, $1 \leq t \leq k$, it follows that they belong to exactly m Q_t's, $1 \leq t \leq k$. ∎

Claim 6: Processor M_2 executes the chains $H_{3m}, H_{3m-1}, \ldots, H_2, H_1$ in decreasing order of their lengths.
Proof. By the construction of the T_t trees, it is clear that a chain H_s of a Q_t, $1 \leq t \leq k$, can start executed on M_2 only if the chain H_{s+1} of the corresponding path P_t has finished its execution on M_1 before c time units. We call the execution of such a pair of chains H_{s+1}, H_s executed on processors M_1 and M_2, respectively, a *stage* s of the schedule.
Clearly, processor M_2 executes the chains of Claim 5 in $3m$ stages. In order to prove this claim, we shall argue on the potential work of both processors in each stage s, i.e. the number of tasks and, hence, the chains that can execute in each stage s. Let us denote by x_s (resp. y_s) the potential work (number of tasks) of processor M_2 (resp. M_1) in stage s; by x_0 we denote the number of tasks of Q_0 initially executed on M_2.
By Claim 4, processor M_2 starts by executing at least x_0 tasks of Q_0, $H_{3m+1}-k \leq x_0 \leq H_{3m+1}$. Therefore, what M_1 can execute in the first stage, in order to prepare some work for M_2 without leaving processor M_2 idle for more than k time slots, is at most $y_1 \leq x_0 \leq H_{3m+1}$ tasks. Then, processor M_2 can execute at most $x_1 \leq H_{3m}$ tasks. Similarly, in the second stage M_1 can execute at most $y_2 \leq x_1 \leq H_{3m}$ tasks and processor M_2 at most $x_2 \leq H_{3m-1}$ tasks. In general, the potential work of processor M_2 in stage s is $x_s \leq H_{3m-s+1}$, $1 \leq s \leq 3m$. Summing up these inequalities we obtain, $\sum_{s=1}^{3m} x_s \leq \sum_{s=1}^{3m} H_{3m-s+1} = \sum_{s=1}^{3m} H_s$.
Let $W = \sum_{s=1}^{3m} H_s$. By Claim 5, W is the amount of work (number of tasks) that should be executed on M_2 in stages 1 to $3m$. Assume that processor M_2, at some stage s, executes a chain of length less than H_{3m-s+1}, i.e. $x_s < H_{3m-s+1}$. In this case, the chain H_{3m-s+1} will never can executed on M_2 without creating an idle period of at least a time units on M_2. Note that this idle period can not be filled up with chains G', since $a >> mG'$. Therefore, M_2 will never have the

sum of the potential work in all $3m$ stages equal to W without being idle for more than $c+1+k$ time units in total, a contradiction to Claim 2. Hence, each one of the inequalities $x_s \leq H_{3m-s+1}$, $1 \leq s \leq 3m$, should be an equality, and this completes the proof of the claim. ∎

By Claims 5 and 6, it follows that in order to execute on processor M_2 $\sum_{i=1}^{3m+1} H_i + mG'$ tasks, X3C1 must have a solution F^* corresponding to the m Q_t's, $1 \leq t \leq k$, that processor M_2 has to execute. Thus we can state the following theorem.

Theorem 1 *Finding an optimal schedule for $P2 \mid$ binary trees, $p_j = 1$, $c_{ij} = c \mid C_{max}$ is NP-hard in the strong sense.*

3 A polynomial time algorithm for complete binary trees

In this section, we prove that our problem becomes polynomial when the task graph is a complete binary (out-)tree. A complete binary tree of height h, containing $n = 2^h - 1$ nodes (tasks) is denoted by T_h.
By $x_h, x_{h-1}, ..., x_2, x_1$ we denote the leftmost path of a tree from its root (x_h) to its leftmost leaf (x_1) and by x'_i, $1 \leq i \leq h-1$, we denote the brother of node x_i. By T_i, T'_i we denote the (sub)trees rooted at nodes x_i and x'_i, respectively.
By convention we assume that M_1 executes the root of the tree. A schedule is called *balanced* if $|C_1 - C_2| \leq 1$.
Towards to an optimal schedule construction we consider the number of communications that can be introduced, by an optimal schedule, in a path from the root of T_h to a leaf. Roughly speaking we distinguish between three cases depending on the magnitude of communication delay c with respect to h:
(i) For "large" c's (i.e. when $c \geq 2^h - 1 - h$) no communication is allowed, i.e. the sequential schedule is the optimal one.
(ii) For "medium" c's (i.e. when $2^{h-2} \leq c < 2^h - 1 - h$) only one way communications, i.e. from M_1 to M_2, are allowed.
(iii) For "small" c's (i.e. when $c \leq 2^{h-2} - 1$) two ways communications, i.e. from M_1 to M_2 and from M_2 to M_1, are allowed.

3.1 Large communications

Lemma 1 *For $c \geq 2^h - 1 - h$, the sequential schedule is optimal.*

Proof. If $c \geq 2^h - 1 - h$, then any one way schedule introduces at least a communication in a path from the root to a leaf of T_h. Such a schedule is of length at least $h + c \geq h + 2^h - 1 - h \geq 2^h - 1 = n$. ∎

3.2 A lower bound for small and medium communications

For $c < 2^h - h - 1$ any optimal schedule uses both processors. A lower bound, L, on the length of an optimal schedule can be derived by considering a balanced schedule. In any schedule processor M_2 is idle for at least $c+1$ time slots before executing any node, and therefore:

$$\text{OPT} \geq L = \left\lceil \frac{c+1+2^h-1}{2} \right\rceil = 2^{h-1} + \left\lceil \frac{c}{2} \right\rceil .$$

3.3 Small communications

For the case of small communication delays let us consider a simple schedule in which processor M_1 executes the node x_h (the root of the tree T_h) and the (nodes of the) tree T'_{h-1}, while processor M_2 executes the tree T_{h-1}. Since processor M_2 can start executing the tree T_{h-1} at time $c+1$, this schedule is obviously unbalanced and $C_2 - C_1 = c$. This schedule can be transformed to an eventually balanced one as following: A subset of the trees T'_i, $1 \leq i \leq h-2$, such that the sum of their nodes equals to $\lfloor \frac{c}{2} \rfloor$ are moved from M_2 to be executed on M_1 (after the execution of the node x_h and the tree T'_{h-1}). Furthermore, in order to avoid as far as possible idle slots on processor M_1, processor M_2 executes first (i.e. as soon as possible) the nodes of the path $x_{h-1}, x_{h-2}, ..., x_{h_k+1}$ and then the rest of the nodes of the tree T_{h-1}.
Let us assume that the subtrees $T'_{h_1}, T'_{h_2}, ..., T'_{h_{k-1}}, T'_{h_k}$, where $h-1 > h_1 > h_2 > \ldots > h_{k-1} \geq h_k \geq 1$, are executed on M_1. Each such tree contains $2^{h_i} - 1$ nodes and what we essentially need is to express the quantity $\lfloor \frac{c}{2} \rfloor$ as a sum of terms each one equal to a power of two minus one, i.e. $\lfloor \frac{c}{2} \rfloor = (2^{h_1}-1) + (2^{h_2}-1) + ... + (2^{h_{k-1}}-1) + (2^{h_k}-1)$, $h-1 > h_1 > h_2 > .... > h_{k-1} \geq h_k \geq 1$.
A small problem here is that it is possible to have $h_{k-1} = h_k$ and we will need two trees of height h_k to be executed on M_1. This problem can be easily resolved by executing on M_2 only the part $x_{h-1}, x_{h-2}, ..., x_{h_k+1}$ of the leftmost path of T_h and both the trees T_{h_k} and T'_{h_k} on M_1.

The algorithm below constructs such a schedule which we call *two-ways schedule* since it involves communications in both directions from M_1 to M_2 and from M_2 to M_1.

Algorithm *TWO-WAYS* (h, c);
(1) Find integers $h_1, h_2, ..., h_{k-1}, h_k$, such that $h-1 > h_1 > h_2 > .... > h_{k-1} \geq h_k \geq 1$, and $\lfloor \frac{c}{2} \rfloor = (2^{h_1}-1) + (2^{h_2}-1) + ... + (2^{h_{k-1}}-1) + (2^{h_k}-1)$;
(2) Schedule on M_1, as soon as possible, the node x_h and the tree T'_{h-1};
(3) Schedule on M_2, as soon as possible, the path $x_{h-1}, x_{h-2}, ..., x_{h_k+1}$;
(4) Schedule on M_1, as soon as possible, the trees $T'_{h_1}, T'_{h_2}, ..., T'_{h_k}$ one after the other;
(5) **If** $h_k = h_{k-1}$ **then** schedule on M_1, as soon as possible, the tree T_{h_k};
(6) Schedule on M_2 as soon as possible the remaining nodes of the tree T_{h-1};

Lemma 2 *For $c \leq 2^{h-2} - 1$, the algorithm* TWO-WAYS *constructs an optimal schedule of length L, within time $O(h)$.*

Proof.
Optimality: By its construction the schedule derived by algorithm *TWO-WAYS* is always feasible. In such a schedule, processor M_2 executes $(2^{h-1}-1)-\lfloor\frac{c}{2}\rfloor$ nodes of tree T_{h-1} (the remaining $\lfloor\frac{c}{2}\rfloor$ nodes of tree T_{h-1} are executed on M_1).

Processor M_2 starts executing node x_{h-1} at time $c+1$ and after this time it remains always busy until time $C_2=(1+c)+(2^{h-1}-1)-\lfloor\frac{c}{2}\rfloor=2^{h-1}+\lceil\frac{c}{2}\rceil=L$.

Processor M_1 cannot start executing tree T'_{h_1} earlier than both: (i) the time at which the execution of T'_{h-1} is completed, that is time $t_1=2^{h-1}$, and (ii) c time units after the completion of the execution of node x_{h_1+1} on M_2, that is time $t_2=(1+c)+(h-h_1-1)+c=2c+h-h_1$. Therefore, the execution of tree T'_{h_1} cannot start on M_1 earlier than time $\max\{t_1,t_2\}$. After that time processor M_1 remains always busy, since the execution of each tree T'_{h_i}, $2\leq i\leq k$ can start at the time at which the execution of tree $T'_{h_{i-1}}$ is completed. Therefore, $C_1=\max\{t_1,t_2\}+\lfloor\frac{c}{2}\rfloor$.
Consider first the case $c=2^{h-2}-1$. By the definition of h_1 as the maximum integer such that $2^{h_1}-1\leq\lfloor\frac{c}{2}\rfloor$, it follows that $h_1=h-3$. Therefore, $t_2=2c+h-h_1=2(2^{h-2}-1)+3=2^{h-1}+1>t_1$. Thus, $C_1=\max\{t_1,t_2\}+\lfloor\frac{c}{2}\rfloor=t_2+\lfloor\frac{c}{2}\rfloor=2^{h-1}+1+\lfloor\frac{c}{2}\rfloor=2^{h-1}+\lceil\frac{c}{2}\rceil=L$.
Consider next the case $c\leq 2^{h-2}-2$. By the definition of h_1 again, it follows that $t_2=2c+h-h_1$ is a strictly increasing function of c. By the previous case, if $c=2^{h-2}-1$, then $t_2=2^{h-1}+1$, and, therefore, for $c\leq 2^{h-2}-2$, it follows that $t_2<2^{h-1}+1\leq 2^{h-1}=t_1$. Thus, $C_1=\max\{t_1,t_2\}+\lfloor\frac{c}{2}\rfloor=t_1+\lfloor\frac{c}{2}\rfloor=2^{h-1}+\lfloor\frac{c}{2}\rfloor\leq 2^{h-1}+\lceil\frac{c}{2}\rceil=L$.

Therefore, for $c\leq 2^{h-2}-1$, the length of the schedule derived by algorithm *TWO-WAYS* is $\max\{C_1,C_2\}=L$, and this completes the optimality proof.

Complexity analysis: Step (1) of the algorithm can be easily implemented within $O(k)=O(\log\lfloor\frac{c}{2}\rfloor)$ time, that is $O(h)$, for $c\leq 2^{h-1}-1$. Steps 3, 4 and 6 can also implemented within $O(k)=O(\log h)$, steps 2 and 5 within $O(1)$ time, and the complexity of the algorithm follows. ∎

3.4 Medium communications

We consider finally the remaining case of "medium" c's, that is $2^{h-2}\leq c\leq 2^h-h-1$. For this case we prove first a lower bound on the length L_T of any two-ways algorithm.

Corollary 1 *For $c\geq 2^{h-2}$, any two-ways algorithm derives a schedule of length $L_T\geq L+2$.*

Proof. Consider first the *TWO-WAYS* algorithm. As in the proof of Lemma 2, it still holds that $C_2=L$, $C_1=\max\{t_1,t_2\}+\lfloor\frac{c}{2}\rfloor$ and $t_1=2^{h-1}$. Moreover, if $c=2^{h-2}$, then $h_1=h-3$, and $t_2=2(2^{h-2})+h-h_1=2^{h-1}+3$. Hence, for $c\geq 2^{h-1}$, $t_2\geq 2^{h-1}+3>t1$, and

$C_1 = \max\{t_1, t_2\} + \lfloor \frac{c}{2} \rfloor = t_2 + \lfloor \frac{c}{2} \rfloor \geq 2^{h-1} + 3 + \lfloor \frac{c}{2} \rfloor \geq 2^{h-1} + \lceil \frac{c}{2} \rceil + 2 = L + 2$. Therefore, the length of the derived schedule is $\max\{C_1, C_2\} = C_1 \geq L + 2$.

Compare next the *TWO-WAYS* algorithm to any other two-ways algorithm A. The *TWO-WAYS* algorithm derives a balanced schedule with the smallest possible number of idle time slots. Algorithm A violates some step(s) of the *TWO-WAYS* one and, therefore, derives either an unbalanced schedule or one that has more idle time slots than the schedule constructed by the *TWO-WAYS* algorithm or both. Hence, algorithm A will derive a longer schedule than the *TWO-WAYS* algorithm. ∎

It is clear by Corollary 1 that the magnitude of medium c's is too long for the *TWO-WAYS* algorithm to be optimal. In this case an eventually balanced scheduling, that contains communications only from M_1 to M_2, can be constructed as following:
Processor M_1 starts by executing (a part of) the path $x_h, x_{h-1}, ..., x_2, x_1$ in consecutive time units. In order to produce a balanced schedule, processor M_2 will execute a subset of the trees T'_i, $1 \leq i \leq h-1$.
Let us assume that trees $T'_{h_1}, T'_{h_2}, ..., T'_{h_{k-1}}, T'_{h_k}$, with $h > h_1 > h_2 > ... > h_{k-1} \geq h_k \geq 1$, are executed on M_2 in order to produce a balanced schedule. The earliest time at which tree T'_{h_1} can start its execution on M_2 is c time units after the completion of node x_{h_1+1} on M_1, that is not earlier than time $1 + c + (h - h_1 - 1)$. Therefore, such a schedule, where processor M_2 starts with $h - h_1 + c$ idle time slots, is of length

$$L_1 = \lceil \tfrac{2^h - 1 + h - h_1 + c}{2} \rceil.$$

To define h_1 we consider a schedule where only the tree T'_{h_1} is executed on M_2. Then, $C_2 = h - h_1 + c + 2^{h_1} - 1$ and $C_1 = (2^h - 1) - (2^{h_1} - 1)$ and in order to produce finally a balanced schedule, h_1 should be the maximum positive integer such that $C_2 - C_1 \leq 1$, that is

$$h_1 = \max_{1 \leq h_i \leq h-1} \{h_i \mid 2^{h_1+1} - h_i \leq 2^h - h - c + 2\}.$$

Therefore, a balanced schedule can be constructed by selecting the trees $T'_{h_i}, 1 \leq i \leq k$, such that the number of their nodes is equal to $L_1 - (h - h_1 + c)$.
The algorithm below constructs such a schedule which we call *one-way schedule*.

Algorithm *ONE-WAY* (h, c);
(1) $h_1 = \max_{1 \leq h_i \leq h-1} \{h_i \mid 2^{h_1+1} - h_i \leq 2^h - h - c + 2\}$;
(2) $L_1 = \left\lceil \frac{2^h - 1 + c + h - h_1}{2} \right\rceil$;
(3) Find integers $h_2, ..., h_{k-1}, h_k$, $h > h_1 > h_2 > ... > h_{k-1} \geq h_k \geq 1$,
such that $L_1 - (h - h_1 + c) = (2^{h_1} - 1) + (2^{h_2} - 1) + ... + (2^{h_{k-1}} - 1) + (2^{h_k} - 1)$;
(4) Schedule on M_1 as soon as possible the path $x_h, x_{h-1}, ..., x_{h_k+1}$;
(5) Schedule on M_2, as soon as possible, the trees $T'_{h_1}, T'_{h_2}, ..., T'_{h_k}$ one after the other;
(6) **If** $h_k = h_{k-1}$ **then** schedule on M_2 as soon as possible the tree T_{h_k};
(7) Schedule on M_1 as soon as possible the remaining nodes of the tree T_h;

Lemma 3 *For $2^{h-2} \leq c \leq 2^h - h - 1$, the algorithm* ONE-WAY *constructs an optimal schedule of length L_1, within time $O(h)$.*

Proof. We prove first that the algorithm *ONE-WAY* derives a feasible schedule of length L_1. Nodes $x_h, x_{h-1}, ..., x_{h_k+1}$ are executed on M_1 in the first $k-1$ time slots. Tree T'_{h_1} starts its execution, on M_2, at time $h - h_1 + c$. Each tree T'_{h_i}, $2 \leq i \leq k$ can start its execution immediately after the completion of the nodes of the tree $T'_{h_{i-1}}$. Therefore, no idle time slot is introduced on any processor after the time it started executing some node and a feasible schedule of length L_1 is derived.
We prove next that a schedule of length L_1 is optimal. Any other one-way algorithm, violating some step(s) of *ONE-WAY* algorithm, will introduce more idle time slots in the schedule constructed than *ONE-WAY* does, resulting on a schedule longer than L_1. To complete the optimality proof we compare L_1 with the length L_T of any two-ways schedule for these values of c:
(i) If $c \leq 2^{h-1}$, then $h_1 = h - 2$ and $L_1 = \lceil \frac{2^h - 1 + h - h_1 + c}{2} \rceil = \lceil \frac{2^h - 1 + 2 + c}{2} \rceil = \lceil \frac{2^h + 1 + c}{2} \rceil = L + \mathbf{1}(c \bmod 2 = 0)$. Using Corollary 1, $L_T \geq L + 2 > L + \mathbf{1}(c \bmod 2 = 0) = L_1$.
(ii) If $2^{h-1} < c \leq 2^h - h - 1$, then $L_T \geq 2c + h$, since there is at least one path in T_h with two communications introduced. That is, $L_T \geq 2c + h = 2^{h-1} - 1 + h + c > \lceil \frac{2^h - 1 + h - h_1 + c}{2} \rceil = L_1$.
This completes the optimality proof.
Similarly with Lemma 2, the complexity of the algorithm is $O(k) = O(\log L_1) = O(h)$. ∎

Remark By the proof of Lemma 3, algorithm *ONE-WAY*, for every odd c, $c \leq 2^{h-1}$, and for $h_1 = h - 2$, gives an optimal schedule of length L. Combining this observation with Lemma 2 it follows that for odd c, $c \leq 2^{h-2} - 1$ both *TWO-WAYS* and *ONE-WAY* algorithms derive an optimal schedule of length L.

Combining Lemmas 1, 2 and 3 we obtain the next theorem:

Theorem 2 *A complete binary tree of height h, with communication delay c, can be scheduled optimally on two processors within $O(h)$ time.*

Remarks
(i) The complexity of the algorithms is obtained under the assumption that a complete binary tree can be scheduled sequentially within $O(1)$ time.
(ii) The polynomial algorithm, can be easily generalized to k-ary complete trees.

Acknowledgements
The authors thanks the anonymous referees for their valuable suggestions and comments that significantly improved the original submission.

References

[1] H. Ali, H. El-Rewini, *The time complexity of scheduling interval orders with communication is polynomial,* Parallel Processing Letters **3** (1) (1993) 53-58.

[2] E. Bampis, A. Giannakos, J.-C. König, *On the complexity of scheduling with Large communication delays,* Europ. Journal of Operational Research **94** (1996) 252-260.

[3] P. Chrétienne, C. Picouleau, *Scheduling with communication delays: A survey*, In *Scheduling Theory and Its Applications*, P. Chrétienne et al. (Eds.), J. Wiley, 1995.

[4] E. G. Coffman Jr., R. L. Graham, *Optimal scheduling for two-processor systems*, Acta Informatica **1** (1972) 200-213.

[5] H. Jung, L. Kirousis, P. Spirakis, *Lower bounds and efficient algorithms for multiprocessor scheduling of DAGs with communication delays,* Information and Computation **105** (1993) 94-104.

[6] A. Jakoby and R. Reischuk, *The complexity of scheduling problems with communication delays for trees,* In Proc. Scandinavian Workshop on Algorithm Theory, (SWAT'92), Springer Verlag LNCS-621 (1992) 165-177.

[7] L. Finta, Z. Liu, I. Milis, E. Bampis, *Scheduling UET-UCT series parallel graphs on two processors,* Theoretical Computer Science **162** (2) (1996) 323-340.

[8] M. Fujii, T. Kasami, K. Ninomiya, *Optimal sequencing of two equivalent processors,* SIAM J. App. Math. **17** (4) (1969) 784-789.

[9] L. Gao, A. L. Rosenberg and R. K. Sitaraman, *Optimal architecture-independent scheduling of fine-grain tree-sweep computations,* In Proc. 7^{th} IEEE Symposium on Parallel and distributed Processing (1995) 620-629.

[10] M.R. Garey, D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-completeness*, Ed. Freeman, 1979.

[11] R. L. Graham, E. L. Lawler, J. K. Lenstra, K. Rinnooy Kan, *Optimization and approximation in deterministic scheduling: A survey*, Ann. Disc. Math., **5** (1979) 287-326.

[12] F. Guinand, D. Trystram, *Optimal scheduling of UECT trees on two processors*, Technical Report APACHE RR-93-03, IMAG, Grenoble, 1993.

[13] J. K. Lenstra, M. Veldhorst and B. Veltman, *The complexity of scheduling trees with communication delays,* Journal of Algorithms, **20** (1) (1996) 157-173.

[14] M. G. Norman, S. Pelagatti, P. Thanisch, *On the complexity of scheduling with communication delay and contention,* Parallel Processing Letters, **5** (3) (1995) 331-341.

[15] C. Papadimitriou, M. Yannakakis, *Towards an architecture-independent analysis of parallel algorithms,* SIAM J. on Computing, **2** (1990) 322-328.

[16] C. Picouleau, *Etude des problèmes d'optimisation dans les systèmes distribués*, Ph.D. Thesis, Université Pierre et Marie Curie, France, 1992.

[17] V. J. Rayward-Smith, *UET Scheduling with unit interprocessor communication delays*, Disc. Appl. Math., **18** (1987), 55-71.

[18] T. Varvarigou, V. P. Roychowdhury, T. Kailath, E. Lawler, *Scheduling in and out forests in the presence of communication delays* IEEE Trans. on Parallel and Distributed Systems, **7** (10) (1996) 1065-1074.

[19] M. Veldhorst, *A linear time algorithm to schedule trees with communication delays optimally on two machines*, Technical Report COSOR 93-07, Dep. of Math. and Comp. Sci., Eindhoven Univ. of Technology, 1993.

[20] B. Veltman, *Multiprocessor scheduling with communication delays*, Ph.D. Thesis, CWI-Amsterdam, (1993).

[21] B. Veltman, B. J. Lageweg, J. K. Lenstra, *Multiprocessor scheduling with communication delays*, Parallel Computing, **16** (1990) 173-182.