Scheduling outtrees of height one in the LogP model*

Jacques Verriet

Department of Computer Science, Utrecht University,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands.
E-mail: jacques@cs.uu.nl

Abstract

The LogP model is a model of parallel computation that characterises a parallel computer system by four parameters: the latency L, the overhead o, the gap g and the number of processors P. We study the complexity of scheduling fork graphs in the LogP model. It will be proved that constructing minimum-length schedules for fork graphs in the LogP model is a strongly NP-hard optimisation problem. We also present a polynomial-time algorithm that constructs schedules that are at most twice as long as minimum-length schedules. Moreover, we prove that if all tasks of a fork graph have the same execution length, then a minimum-length schedule can be constructed in polynomial time.

1 Introduction

In recent years, a great variety of parallel computer systems have been developed. Because of this variety, it is very difficult to construct a model that characterises all aspects of communication of a parallel computer system. A model of parallel computation can be used to focus on those aspects of communication in a parallel computer system that have a large impact on the length of a schedule, an execution of a computer program. A good model of parallel computation helps to understand the essence of the problem of multiprocessor scheduling with communication.

The PRAM [11] is the most common model of parallel computation. A PRAM consists of a collection of processors that execute a computer program in a synchronous manner; processors communicate by writing and reading in global memory. The PRAM model does not capture the complexity of communication in the execution of computer programs: a communication step takes the same amount of time as a local computation step, whereas, in a real parallel computer system, a communication step is far more time consuming. There are several PRAM-based models of parallel computation that include aspects of communication in a real parallel computer system, such as latency [2, 3, 22], memory contention [15, 14] and asynchrony [6, 13].

Most models of parallel computation include only one or two aspects of communication in real parallel computer systems, but some include more aspects. These models are all architecture independent and characterise the execution of computer programs in a real parallel

*This research was partially supported by ESPRIT Long Term Research Project 20244 (project ALCOM IT: *Algorithms and Complexity in Information Technology*).

computer system by a small number of parameters. Such models are the BSP model [23], the LogP model [9] and the Postal model [4]. In this paper, we will consider the LogP model that provides more control over the machine resources than the BSP model and is more general than the Postal model.

The LogP model captures the characteristics of communication in a real parallel computer system using four parameters.

1. The *latency* L is an upper bound on the time required to send a unit-length message from one processor to another via the communication network. The latency depends on the diameter of the network topology.

2. The *overhead* o is the amount of time during which a processor is involved in sending or receiving a message consisting of one word. During this time, a processor cannot perform other operations.

3. The *gap* g is the minimum length of the delay between the starting times of two consecutive message transmissions or two consecutive message receptions on the same processor. $\frac{1}{g}$ is the communication *bandwidth* available for each processor.

4. P is the *number of processors*.

We will assume that L, o and g are non-negative integers and that $P \in \{2, 3, \ldots, \infty\}$.

In addition, Culler et al. [9] make the following assumptions. The communication network is assumed to be of finite capacity: at each time at most $\lceil \frac{L}{g} \rceil$ messages can be in transit from or to any processor. If a processor attempts to send a message that causes such a bound to be exceeded, then this processor stalls until the message can be sent without exceeding the bound of $\lceil \frac{L}{g} \rceil$ messages. Moreover, the time needed to transfer a message from one processor to another is assumed to be exactly L time units: any message arrives at its destination processor exactly L time units after it has been submitted to the communication network by its source processor.

The communication between processors in the LogP model is modelled as follows. Consider two different processors p_1 and p_2. Assume that data has to be transferred from processor p_1 to processor p_2 and that this data is contained in two messages. Then two messages must be sent from processor p_1 to processor p_2. Figure 1 shows the communication between processors p_1 and p_2. The *send operations* are represented by s_1 and s_2; r_1 and r_2 are the *receive operations* corresponding to s_1 and s_2, respectively.

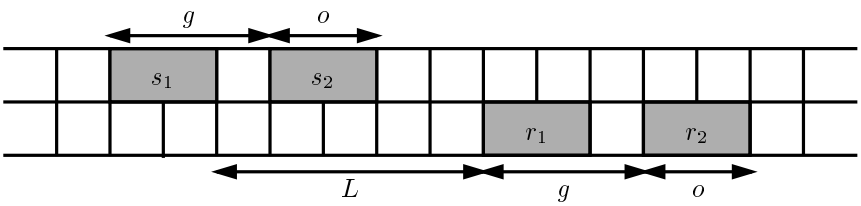

Figure 1: Communication between two processors in the LogP model

It takes o units of time to submit the first message to the communication network. Exactly L time units after this message is submitted it can be received by processor p_2. The second

message cannot be sent immediately after the first: there must be a delay of at least g time units between the starting times of two consecutive send operations on the same processor. The second message can be received L time units after it has been sent. Note that since both messages are received exactly L time units after they have been sent, there is a delay of g time units between the starting times of the receive operations on processor p_2.

Like for many other models of parallel computation, little is known about scheduling in the LogP model. A few algorithms have been presented that construct schedules in the LogP model for common computer programs. These programs include sorting [1, 10], broadcast [18] and the Fast Fourier Transform [8].

In addition, Löwe and Zimmermann [20, 25] presented an algorithm that constructs schedules for communication structures of PRAMs on an unrestricted number of processors. The length of these schedules is at most $1 + \frac{1}{\gamma(G)}$ times the length of a minimum-length schedule, where $\gamma(G)$ is the grain size of G. Löwe et al. [21] proved the same result for a generalisation of the LogP model. Moreover, Löwe and Zimmermann [20] presented an algorithm that constructs schedules that are at most twice as long as a minimum-length schedule plus the duration of the sequential communication operations.

Kort and Trystram [19] presented three algorithms for scheduling join graphs. They proved that if g equals o and all task lengths or all message lengths are equal, then a minimum-length schedule for a join graph on an unrestricted number of processors can be constructed in polynomial time. In addition, Kort and Trystram [19] showed that if all tasks have the same length and this length is at least $\max\{g, 2o + L\}$, then a minimum-length schedule for a join graph on two processors can be constructed in linear time.

This paper is concerned with the problem of constructing minimum-length schedules for fork graphs in the LogP model. It is proved that constructing minimum-length schedules for a fork graph on an unrestricted number of processors is a strongly NP-hard optimisation problem. A polynomial-time algorithm is presented that constructs schedules for fork graphs on P processors that are at most twice as long as a minimum-length schedule on P processors. In addition, it is shown that if all task lengths are equal, then a minimum-length schedule for a fork graph on P processors can be constructed in polynomial time.

2 Preliminaries

In this paper, we consider the problem of scheduling a computer program in the LogP model. Such a program is represented by a triple (G, μ, c), such that $G = (V, E)$ is a precedence graph, $\mu : V \to \mathbb{Z}^+$ and $c : V \to \mathbb{N}$. An element of V will be called a *task* of G; an element of E an *arc* of G. A task of G corresponds to a task of a computer program; an arc to a *data dependency* between the tasks: if there is an arc from task u_1 to task u_2, then the result of u_1 is needed to execute u_2. Task u has execution length $\mu(u)$: the execution of u on a processor takes $\mu(u)$ time. If the result of u must be transferred to another processor, then $c(u)$ messages need to be sent to this processor. A LogP scheduling instance is represented by a tuple (G, μ, c, L, o, g, P), where (G, μ, c) represents a computer program and (L, o, g, P) contains the parameters of the LogP model.

Consider an instance (G, μ, c, L, o, g, P). Let u_1 and u_2 be two tasks of G. If there is an arc from u_1 to u_2, then u_2 is called a *child* of u_1 and u_1 a *parent* of u_1. If there is a directed path from u_1 to u_2, then u_2 is called a *successor* of u_1 and u_1 a *predecessor* of u_2. This is

denoted by $u_1 \prec_G u_2$.

Consider an arc (u_1, u_2) of G. Assume u_1 and u_2 are scheduled on different processors. Assume u_1 is executed on processor p_1 and u_2 on processor p_2. Then $c(u_1)$ messages $m_{u_1,1}, \ldots, m_{u_1,c(u_1)}$ have to be sent from processor p_1 to processor p_2. Sending message $m_{u_1,i}$ to processor p_2 will be represented by the *send operation* $s_{u_1,p_2,i}$. This send operation must be executed on processor p_1. The reception of message $m_{u_1,i}$ is represented by a *receive operation* $r_{u_1,p_2,i}$ that must be executed by processor p_2.

We will define two sets $S(G, P, c)$ and $R(G, P, c)$ containing the send and the receive operations, respectively. $S(G, P, c)$ contains the send operations $s_{u,p,i}$, such that u is a task of G, $p \in \{1, \ldots, P\}$ is a processor and $i \in \{1, \ldots, c(u)\}$ is the index of a message of u. The set $R(G, P, c)$ contains the receive operations $r_{u,p,i}$, such that u is a task of G, $p \in \{1, \ldots, P\}$ and $i \in \{1, \ldots, c(u)\}$. Let $C(G, P, c)$ be the union of $S(G, P, c)$ and $R(G, P, c)$, the set of *communication operations*. Each communication operation u in $C(G, P, c)$ has *length* $\mu(u) = o$.

A *schedule* for an instance $(G = (V, E), \mu, c, L, o, g, P)$ is a pair of functions (σ, π), such that $\sigma : V \cup C(G, P, c) \to \mathbb{N} \cup \{\perp\}$ and $\pi : V \cup C(G, P, c) \to \{1, \ldots, P\} \cup \{\perp\}$. σ assign a starting time and a processor to every task of G and every communication operation in of $C(G, P, c)$. The value $\perp$ denotes the starting time and processor of communication operations that are not scheduled.

Definition 2.1. A schedule (σ, π) for (G, μ, c, L, o, g, P) is called a *feasible schedule* for (G, μ, c, L, o, g, P) if

1. for all tasks u of G, $\sigma(u) \neq \perp$ and $\pi(u) \neq \perp$;
2. for all elements u_1 and u_2 of $V(G) \cup C(G, P, c)$, if $\pi(u_1) = \pi(u_2) \neq \perp$, then $\sigma(u_1) + \mu(u_1) \leq \sigma(u_2)$ or $\sigma(u_2) + \mu(u_2) \leq \sigma(u_1)$;
3. for all tasks u_1 and u_2 of G, if $u_1 \prec_G u_2$, then $\sigma(u_1) + \mu(u_1) \leq \sigma(u_2)$;
4. for all tasks u_1 and u_2 of G, if u_2 is a child of u_1 and $\pi(u_1) \neq \pi(u_2)$, then, for all $i \leq c(u_1)$, $\pi(s_{u_1,\pi(u_2),i}) = \pi(u_1)$, $\pi(r_{u_1,\pi(u_2),i}) = \pi(u_2)$, $\sigma(s_{u_1,\pi(u_2),i}) \geq \sigma(u_1) + \mu(u_1)$, $\sigma(r_{u_1,\pi(u_2),i}) = \sigma(s_{u_1,\pi(u_2),i}) + o + L$ and $\sigma(u_2) \geq \sigma(r_{u_1,\pi(u_2),i}) + o$;
5. for all send operations s_1 and s_2 in $S(G, P, c)$, if $\pi(s_1) = \pi(s_2) \neq \perp$, then $\sigma(s_1) + g \leq \sigma(s_2)$ or $\sigma(s_2) + g \leq \sigma(s_1)$;
6. for all receive operations r_1 and r_2 in $R(G, P, c)$, if $\pi(r_1) = \pi(r_2) \neq \perp$, then $\sigma(r_1) + g \leq \sigma(r_2)$ or $\sigma(r_2) + g \leq \sigma(r_1)$; and
7. for all tasks u of G and all processors p, if no child of u is scheduled on processor p or $p = \pi(u)$, then $\sigma(s_{u,p,i}) = \perp$ and $\pi(r_{u,p,i}) = \perp$ for all $i \in \{1, \ldots, c(u)\}$.

The first constraint states that all tasks of G have to be executed. The second and third ensure that a processor does not execute two tasks at the same time and that a task is be scheduled after its predecessors. The fourth states that messages have to be sent if a task and one of its children are scheduled on different processors. Moreover, it states that a message must be received exactly L time units after it has been submitted to the communication network. The fifth and sixth constraint ensure that there is a delay of at least g time units between two consecutive send or receive operations on the same processor. Note that there

need not be a delay between a send operation and a receive operation on the same processor. The last constraint states that some communication operations need not be executed.

In the definition of the LogP model [9], a source processor can send a message to a destination processor, unless the number of messages in transit from the source processor or to the destination processor exceeds $\lceil \frac{L}{g} \rceil$, in which case the source processor stalls. The definition of feasible schedules in the LogP model states that a receive operation must be executed exactly L time units after the corresponding send operation has been completed. So each processor can send at most one message in g consecutive time units and at most one message can be sent to the same processor in g consecutive time units. Hence the number of messages in transit from or to any processor cannot be larger than $\lfloor \frac{L+\max\{o,g\}-1}{\max\{o,g\}} \rfloor \leq \lceil \frac{L-1}{g} \rceil + 1 \leq \lceil \frac{L}{g} \rceil$. So we do not need to consider stalling.

Constructing a schedule for an instance (G, μ, c, L, o, g, P) corresponds to assigning a starting time and a processor to every task of G and every communication operation in $C(G, P, c)$. Hence any algorithm that constructs feasible schedules for instances $(G = (V, E), \mu, c, L, o, g, P)$ uses at least $\Theta(\sum_{u \in V} c(u))$ time. If $c_{\max} = \max_{u \in V} c(u)$ is not bounded by a polynomial in n and $\log \max_{u \in V} \mu(u)$, then such an algorithm cannot have a polynomial time complexity.

In a well-structured computer program, the size of a result of a task is not very large. Hence we may assume that $c_{\max}$ is not exponentially large. In the remainder of this paper, we do not want to focus on the time needed to schedule the communication operations. Hence we will assume that $c_{\max}$ is bounded by a constant. However, the time complexity of the algorithms presented in this paper remains polynomial if $c_{\max}$ is bounded by a polynomial in n and $\log \max_{u \in V} \mu(u)$.

In the remainder of this paper, we will focus on a special class of precedence graphs, the *fork graphs*. A fork graph is a outtree of height one: it consists of a task, the *source*, and its children, the *sinks*. Example 2.2 shows a schedule for a fork graph in the LogP model.

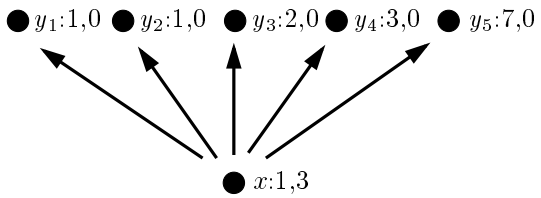

Figure 2: An instance $(G, \mu, c, 2, 1, 2, 2)$

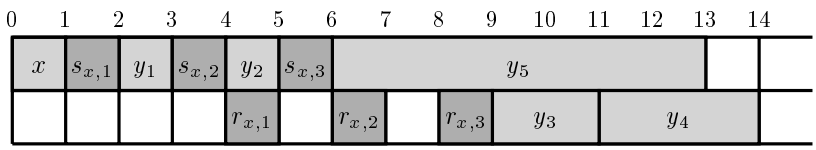

Figure 3: A feasible schedule for $(G, \mu, c, 2, 1, 2, 2)$

Example 2.2. Consider the instance $(G, \mu, c, 2, 1, 2, 2)$ shown in Figure 2. It is not difficult

to see that the schedule shown in Figure 3 is a feasible schedule for $(G, \mu, c, 2, 1, 2, 2)$. Note that y_1 and y_2 are scheduled between the send operations on processor 1. No task can be executed between the receive operations on processor 2, since all three messages are needed to send the result of x to another processor. Although two children of x are executed on the second processor, only three send and receive operations are executed: the result of x has to be sent to the second processor exactly once.

3 An NP-completeness result

In this section, we study the complexity of constructing minimum-length schedules for fork graphs in the LogP model. If the number of processors is restricted, then it is not difficult to prove that this optimisation problem is NP-hard. Using a polynomial reduction from 3PARTITION, it will be shown that constructing minimum-length schedules for fork graphs on an unrestricted number of processors is strongly NP-hard. 3PARTITIONis defined as follows [12].

Problem. 3PARTITION
Instance. A set $A = \{a_1, \ldots, a_{3m}\}$ of positive integers and an integer B, such that $\sum_{i=1}^{3m} a_i = mB$ and $\frac{1}{4}B < a_i < \frac{1}{2}B$ for all $i \in \{1, \ldots, 3m\}$.
Question. Are there pairwise disjoint subsets $A_1, \ldots, A_m$ of A, such that $\sum_{a \in A_j} a = B$ for all $j \in \{1, \ldots, m\}$?

3PARTITION is a well-known strongly NP-complete decision problem [12]. FORK GRAPH SCHEDULING is the following decision problem.

Problem. FORK GRAPH SCHEDULING
Instance. An instance $(G, \mu, c, L, o, g, \infty)$, such that G is a fork graph and an integer D.
Question. Is there a feasible schedule for $(G, \mu, c, L, o, g, \infty)$ of length at most D?

Lemma 3.1 shows the existence of a polynomial reduction from 3PARTITION to FORK GRAPH SCHEDULING. This reduction shows that FORK GRAPH SCHEDULING is a strongly NP-complete decision problem.

Lemma 3.1. *There is a polynomial reduction from* 3PARTITION *to* FORK GRAPH SCHEDULING.

Proof. Let $A = \{a_1, \ldots, a_{3m}\}$ and B be an instance of 3PARTITION. Construct an instance $(G, \mu, c, L, o, g, \infty)$ of FORK GRAPH SCHEDULING as follows. G is a fork graph with source x and sinks $y_1, \ldots, y_{3m}$ and $z_1, \ldots, z_{m+2}$. Let $\mu(x) = 1$, $\mu(y_i) = a_i$ for all $i \in \{1, \ldots, 3m\}$, $\mu(z_1) = 3mB$ and $\mu(z_i) = 3mB + (m + 2 - i)B$ for all $i \in \{2, \ldots, m + 2\}$. Let $c(x) = 1$, $c(y_i) = 0$ for all $i \in \{1, \ldots, 3m\}$ and $c(z_i) = 0$ for all $i \in \{1, \ldots, m + 2\}$. Let $L = 0$, $o = 0$ and $g = B$. In addition, let $D = 4mB + 1$. Now it is proved that there are pairwise disjoint subsets $A_1, \ldots, A_m$ of A, such that $\sum_{a \in A_j} a = B$ for all $j \in \{1, \ldots, m\}$ if and only if there is a feasible schedule for $(G, \mu, c, L, o, g, \infty)$ of length at most D.

($\Rightarrow$) Assume $A_1, \ldots, A_m$ are pairwise disjoint subsets of A, such that $\sum_{a \in A_j} a = B$ for all $j \in \{1, \ldots, m\}$. Then $A_1 \cup \ldots \cup A_m = A$. A schedule (σ, π) for $(G, \mu, c, L, o, g, \infty)$ can be constructed as follows. x starts at time 0 on processor 1. For all $i \in \{2, \ldots, m+2\}$, send operation $s_{x,i,1}$ is executed at time $(i-2)B+1$ on processor 1 and receive operation $r_{x,i,1}$

at time $(i-2)B+1$ on processor i. Sink z_1 is scheduled at time $mB+1$ on processor 1 and sink z_i at time $(i-2)B+1$ on processor i for all $i \in \{2,\ldots,m+2\}$. For all $j \in \{1,\ldots,m\}$, define $Y_j = \{y_i \mid a_i \in A_j\}$. Then $\sum_{y\in Y_j} \mu(y) = B$ for all $j \in \{1,\ldots,m\}$. The tasks of Y_j are scheduled without interruption from time $(j-1)B+1$ to time $jB+1$ on processor 1. Then the sinks $y_1,\ldots,y_{3m}$ are scheduled between the send operations on processor 1 and the sinks $z_1,\ldots,z_{m+2}$ after the communication operations. Hence (σ,π) is a feasible schedule for (G,μ,c,L,o,g,∞). Its length equals $\max_{1\le i\le m+2}(\sigma(z_i)+\mu(z_i))$. z_1 is completed at time $\sigma(z_1)+\mu(z_1) = mB+1+3mB = 4mB+1$. For all $i \in \{2,\ldots,m+2\}$, sink z_i finishes at time $\sigma(z_i)+\mu(z_i) = (i-2)B+1+3mB+(m+2-i)B = 4mB+1$. Hence (σ,π) is a feasible schedule for (G,μ,c,L,o,g,∞) of length at most D.

($\Leftarrow$) Assume (σ,π) is a feasible schedule for (G,μ,c,L,o,g,∞) of length at most D. Then $\pi(z_i) \neq \pi(z_j)$ for all $i \neq j$. So the tasks of G are scheduled on at least $m+2$ processors. Assume x is scheduled at time 0 on processor 1. There is a sink z_i that is scheduled after $m+1$ receive operations. This task cannot start until time $mg+1 = mB+1$. Since $\mu(z_i) \ge 3mB$ for all $i \in \{1,\ldots,m+2\}$, we may assume that z_{m+2} is scheduled at time $mB+1$. Since it starts at time $mB+1$, send operations must be executed at times $(i-2)B+1$ on processor 1 for all $i \in \{2,\ldots,m+2\}$. We may assume that send operation $s_{x,i,1}$ is scheduled at time $(i-2)B+1$ on processor 1 and receive operation $r_{x,i,1}$ at the same time on processor i. Hence we may assume that $\pi(z_{m+2}) = m+2$. The remaining sinks $z_1,\ldots,z_{m+1}$ must be scheduled on processors $1,\ldots,m+1$. Since the length of the sinks $z_2,\ldots,z_{m+1}$ is larger than $3mB$, z_1 must be scheduled on processor 1 at time $mB+1$. Similarly, sink z_i must be scheduled on processor i at time $(i-2)B+1$ for all $i \in \{2,\ldots,m+1\}$. Then all sinks $z_1,\ldots,z_{m+2}$ finish at time $4mB+1$. A sink y_i cannot be executed on processor $j \neq 1$ before sink z_j, because z_j is scheduled immediately after receive operation $r_{x,j,1}$. So sinks $y_1,\ldots,y_{3m}$ are scheduled between the send operations on processor 1. There is a delay of mB time units between the first and last send operation. Since the sum of the length of the sinks $y_1,\ldots,y_{3m}$ equals mB, processor 1 is not idle before time D. No sink y_i can start before a send operation and finish after it. For all $j \in \{2,\ldots,m+1\}$, define $Y_{j-1} = \{y_i \mid (j-2)B+1 \le \sigma(y_i) < (j-1)B+1\}$ and $A_{j-1} = \{a_i \mid y_i \in Y_j\}$. Then the sets A_j are pairwise disjoint and $\sum_{a\in A_j} a = \sum_{y\in Y_j} \mu(y) = B$ for all $j \in \{1,\ldots,m\}$.

□

Lemma 3.1 shows that FORK GRAPH SCHEDULING is a strongly NP-complete decision problem and that constructing minimum-length schedules for fork graphs on an unrestricted number of processors is a strongly NP-hard optimisation problem.

Theorem 3.2. *Constructing minimum-length schedules for instances (G,μ,c,L,o,g,∞), such that G is a fork graph, is a strongly* NP*-hard optimisation problem.*

The reduction presented in the proof of Lemma 3.1 uses the fact that g may exceed o. Using a reduction from PARTITION [12], one can also prove that if $o \ge g$ and $o \ge 1$, then constructing a minimum-length schedule for a fork graph on an unrestricted number of processors is an NP-hard optimisation problem. It is not clear whether constructing a minimum-length schedule for a fork graph on an unrestricted number of processors remains

NP-hard if o and g are bounded by a constant. If both o and g equal zero, then a minimum-length schedule for a fork graph on an unrestricted number of processors can be constructed in polynomial time [5].

4 A 2-approximation algorithm

In this section, a simple 2-approximation algorithm for scheduling fork graphs in the LogP model is presented. It is not difficult to see that in a minimum-length schedule for an instance (G, μ, c, L, o, g, P), such that G is a fork graph, the number of processors on which a task of G is scheduled need not exceed the number of sinks of G. This knowledge is used in this section: for each possible number of processors m, we will construct a schedule for (G, μ, c, L, o, g, P) in which the tasks of G are scheduled on exactly m processors. It will be proved that the shortest of these schedules is at most twice as long as a minimum-length schedule for (G, μ, c, L, o, g, P).

Consider an instance (G, μ, c, L, o, g, P), such that G is a fork graph with source x and sinks $y_1, \ldots, y_n$. There is a minimum-length schedule for (G, μ, c, L, o, g, P) in which the tasks of G are scheduled on at most $\min\{n, P\}$ processors. Let $m \leq \min\{n, P\}$ be a positive integer. A feasible schedule for (G, μ, c, L, o, g, P) will be called an *m-processor schedule* for (G, μ, c, L, o, g, P) if there are exactly m processors on which a task of G is executed. More precisely, a feasible schedule (σ, π) for (G, μ, c, L, o, g, P) is an m-processor schedule for (G, μ, c, L, o, g, P) if $|\{\pi(u) \mid u \text{ is a task of } G\}| = m$.

Consider an instance (G, μ, c, L, o, g, P), such that G is a fork graph with source x and sinks $y_1, \ldots, y_n$, and a positive integer $m \leq \min\{n, P\}$. Algorithm FORK GRAPH SCHEDULING shown in Figure 4 constructs an m-processor schedule for (G, μ, c, L, o, g, P) as follows. The source x of G is scheduled at time 0 on processor 1 and a set of $c(x)$ send and receive operations is scheduled for each of the processors $2, \ldots, m$. To ensure that the constructed schedule is an m-processor schedule, a sink of G is scheduled immediately after the last receive operation on each of these processors. The remaining sinks are scheduled by a straightforward modification of Graham's List scheduling algorithm [16, 17].

Example 4.1. Consider the instance $(G, \mu, c, 2, 1, 2, \infty)$ shown in Figure 5. For this instance, Algorithm FORK GRAPH SCHEDULING constructs the 3-processor schedule shown in Figure 6. First x is scheduled on processor 1 at time 0. The result of x is sent to processors 2 and 3 as early as possible. Sink y_1 is scheduled immediately after the last receive operation on processor 2. Similarly, y_2 is scheduled immediately after the last receive operation on processor 3. The remaining sinks are scheduled after the send operations on processor 1, after y_1 on processor 2, or after y_2 on processor 3.

Now we will prove that Algorithm FORK GRAPH SCHEDULING correctly constructs m-processor schedules for fork graphs.

Lemma 4.2. *Let G be a fork graph with source x and sinks $y_1, \ldots, y_n$. Let $m \leq \min\{n, P\}$ be a positive integer. Let (σ_m, π_m) be the schedule for (G, μ, c, L, o, g, P) constructed by Algorithm* FORK GRAPH SCHEDULING. *Then (σ_m, π_m) is an m-processor schedule for (G, μ, c, L, o, g, P).*

Proof. x is executed at time 0 on processor 1. It is easy to see that all sinks of G are scheduled after x. For all processors $p \in \{2, \ldots, m\}$ and all $j \in \{1, \ldots, c(x)\}$, send operation $s_{x,p,j}$ is

Algorithm FORK GRAPH SCHEDULING
Input. An instance (G, μ, c, L, o, g, P), such that G is a fork graph with source x and sinks $y_1, \ldots, y_n$ and a positive integer $m \leq \min\{n, P\}$.
Output. A m-processor schedule (σ_m, π_m) for (G, μ, c, L, o, g, P).

1. $\sigma_m(x) := 0$
2. $\pi_m(x) := 1$
3. $idle(1) := \mu(x)$
4. **for** $p := 2$ **to** m
5. **do** $idle(p) := 0$
6. **for** $j := 1$ **to** $c(x)$
7. **do** $\sigma_m(s_{x,p,j}) := \mu(x) + ((p-2)c(x) + j - 1)\max\{o, g\}$
8. $\pi_m(s_{x,p,j}) := 1$
9. $idle(1) := \sigma_m(s_{x,p,j}) + o$
10. $\sigma_m(r_{x,p,j}) := \mu(x) + ((p-2)c(x) + j - 1)\max\{o, g\} + L + o$
11. $\pi_m(r_{x,p,j}) := p$
12. $idle(p) := \sigma_m(r_{x,p,j}) + o$
13. $\sigma_m(y_{p-1}) := idle(p)$
14. $\pi_m(y_{p-1}) := p$
15. $idle(p) := \sigma_m(y_{p-1}) + \mu(y_{p-1})$
16. **for** $i := m$ **to** n
17. **do** assume $idle(p) = \min_{1 \leq j \leq m} idle(j)$
18. $\sigma_m(y_i) := idle(p)$
19. $\pi_m(y_i) := p$
20. $idle(p) := idle(p) + \mu(y_i)$

Figure 4: Algorithm FORK GRAPH SCHEDULING

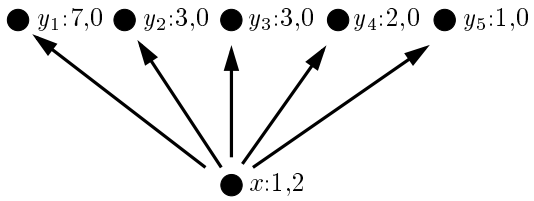

Figure 5: An instance $(G, \mu, c, 2, 1, 2, \infty)$

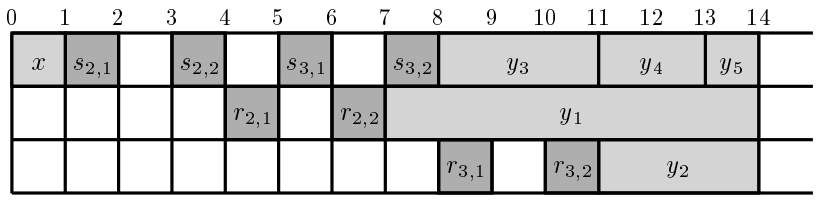

Figure 6: A 3-processor schedule constructed by Algorithm FORK GRAPH SCHEDULING

scheduled on processor 1 at time $\mu(x) + ((p-2)c(x) + j - 1)\max\{o, g\}$ and the corresponding

receive operation $r_{x,p,j}$ on processor p at time $\mu(x)+((p-2)c(x)+j-1)\max\{o,g\}+o+L$. So the send operations are scheduled after x and there is a delay of $\max\{o,g\}$ time units between the starting times of two consecutive send operations or two consecutive receive operations on the same processor. Moreover, there is a delay of exactly L time units between the completion time of a send operation and the starting time of the corresponding receive operation. For all processors $p \in \{2,\ldots,m\}$, a sink of G is scheduled on processor p at the completion time of the last receive operation on processor p. Clearly, the sinks of G are scheduled after all communication operations and no processor executes two tasks at the same time. So (σ_m,π_m) is a feasible schedule for (G,μ,c,L,o,g,P). Because every processor $p \in \{1,\ldots,m\}$ executes at least one task of G, (σ_m,π_m) is an m-processor schedule for (G,μ,c,L,o,g,P). □

The time complexity of Algorithm FORK GRAPH SCHEDULING can be determined as follows. Consider an instance (G,μ,c,L,o,g,P), such that G is a fork graph with n sinks, and a positive integer $m \leq \min\{n,P\}$. Assigning a starting time and a processor to the source of G, $m-1$ sinks of G and the communication operations takes $O(n)$ time. If the processors are stored in a balanced search tree (for instance, a red-black tree [7]) ordered by non-decreasing first idle time, then for each of the remaining $n-m+1$ sinks of G, $O(\log m)$ time is used to determine a starting time and a processor. Hence $O(n\log n)$ time is used to construct an m-processor schedule for (G,μ,c,L,o,g,P).

Lemma 4.3. *For all instances (G,μ,c,L,o,g,P), such that G is a fork graph with n sinks and all positive integers $m \leq \min\{n,P\}$, Algorithm* FORK GRAPH SCHEDULING *constructs a m-processor schedule for (G,μ,c,L,o,g,P) in $O(n\log n)$ time.*

Now it will be proved that the m-processor schedules constructed by Algorithm FORK GRAPH SCHEDULING are at most twice as long as m-processor schedules of minimum length. Let G be a fork graph with source x and sinks $y_1,\ldots,y_n$. Let $m \leq \min\{n,P\}$ be a positive integer. Let (σ_m,π_m) be the m-processor schedule for (G,μ,c,L,o,g,P) constructed by Algorithm FORK GRAPH SCHEDULING. Let ℓ_m be the length of (σ_m,π_m) and ℓ_m^* the length of a minimum-length m-processor schedule for (G,μ,c,L,o,g,P). In any m-processor schedule for (G,μ,c,L,o,g,P), $c(x)$ receive operations have to be executed on $m-1$ processors. Hence if $m \neq 1$, then every m-processor schedule for (G,μ,c,L,o,g,P) has length at least

$$\ell_m^* \geq \mu(x)+((m-1)c(x)-1)\max\{o,g\}+2o+L.$$

Obviously, every 1-processor schedule for (G,μ,c,L,o,g,P) has length at least $\mu(x)+\sum_{i=1}^{n}\mu(y_i)$ and if $m=1$, then Algorithm FORK GRAPH SCHEDULING constructs a schedule of this length. Hence we will assume that $m \geq 2$.

Assume y is a sink of G that finishes at time ℓ_m. Then y has been assigned a starting time and a processor either in Lines 13 and 14 or in Lines 18 and 19 of Algorithm FORK GRAPH SCHEDULING.

Case 1. y has been assigned a starting time and a processor in Lines 13 and 14.

Assume $\pi(y)=p$. Then $p \neq 1$ and y is scheduled immediately after receive operation $r_{x,p,c(x)}$. This receive operation finishes at time $\mu(x)+((p-1)c(x)-1)\max\{o,g\}+2o+L \leq \ell_m^*$. Obviously, $\mu(y) \leq \ell_m^*$. So

$$\begin{aligned}\ell &= \sigma_m(y)+\mu(y)\\ &= (\mu(x)+((p-1)c(x)-1)\max\{o,g\}+2o+L)+\mu(y)\\ &\leq 2\ell_m^*.\end{aligned}$$

Case 2. y has been assigned a starting time and a processor in Lines 18 and 19.

Assume y is scheduled on processor p. If $p = 1$, then y is scheduled after x and the send operations. Otherwise, y is scheduled after sink y_{p-1}. If processor 1 is idle at a time t, such that $\mu(x) + ((m-1)c(x) - 1)\max\{o, g\} + o \leq t < \sigma_m(y)$, then y would have been scheduled at time t on processor 1. Similarly, if a processor $p' \in \{2, \ldots, m\}$ is idle at a time t, such that $\mu(x) + ((p'-1)c(x) - 1)\max\{o, g\} + 2o + L + \mu(y_{p'-1}) \leq t < \sigma_m(y)$, then y would have been scheduled at time t on processor p'. Hence processor 1 is busy from time $\mu(x) + ((m-1)c(x) - 1)\max\{o, g\} + o$ until time $\sigma_m(y)$ and each processor $p' \in \{2, \ldots, m\}$ from time $\mu(x) + ((p'-1)c(x) - 1)\max\{o, g\} + 2o + L + \mu(y_{p'-1})$ until time $\sigma_m(y)$.

No sink of G can be executed before a receive operation on a processor $p \in \{2, \ldots, m\}$. Because the communication operations are executed as early as possible, the idle periods in (σ_m, π_m) on processors $2, \ldots, m$ before the first sink cannot be avoided. Hence the only idle time in (σ_m, π_m) that can be avoided is the idle time between the send operations on processor 1. As a result,

$$\begin{aligned} \ell_m^* &\geq \tfrac{1}{m}(m\sigma_m(y) + \mu(y) - ((m-1)c(x) - 1)(\max\{o, g\} - o)) \\ &= \sigma_m(y) + \tfrac{1}{m}\mu(y) - \tfrac{1}{m}((m-1)c(x) - 1)(\max\{o, g\} - o). \end{aligned}$$

In addition, $\ell_m^* \geq \mu(y)$ and $\ell_m^* \geq \mu(x) + ((m-1)c(x) - 1)\max\{o, g\} + 2o + L$, since the last receive operation on the m^{th} processor cannot be completed before this time. Consequently,

$$\begin{aligned} \ell_m &= \sigma_m(y) + \mu(y) \\ &\leq \ell_m^* + (1 - \tfrac{1}{m})\mu(y) + \tfrac{1}{m}(((m-1)c(x) - 1)(\max\{o, g\} - o)) \\ &\leq \ell_m^* + (1 - \tfrac{1}{m})\ell_m^* + \tfrac{1}{m}\ell_m^* \\ &= 2\ell_m^*. \end{aligned}$$

Consequently, (σ_m, π_m) is at most twice as long as a minimum-length m-processor schedule for (G, μ, c, L, o, g, P).

For each positive integer $m \leq \min\{n, P\}$, Algorithm FORK GRAPH SCHEDULING is used to construct an m-processor schedule (σ_m, π_m) for (G, μ, c, L, o, g, P) of length ℓ_m. Assume (σ_k, π_k) is the shortest of these schedules. Let $\ell^* = \min_{1 \leq m \leq \min\{n,P\}} \ell_m^*$. Assume $\ell^* = \ell_p^*$. Then $\ell_k \leq \ell_p \leq 2\ell_p^* = 2\ell^*$. Hence we have proved the following result.

Theorem 4.4. *There is an algorithm with an $O(n^2 \log n)$ time complexity that constructs feasible schedules for instances (G, μ, c, L, o, g, P), such that G is a fork graph with n sinks, with length at most $2\ell^*$, where ℓ^* is the length of a minimum-length schedule for (G, μ, c, L, o, g, P).*

5 A polynomial special case

In Section 3, it was shown that constructing minimum-length schedules for fork graphs is a strongly NP-hard optimisation problem. In Section 4, a 2-approximation algorithm was presented. In this section, it will be proved that if all sinks of a fork graph have the same task length, then a minimum-length schedule can be constructed in polynomial time.

Let G be a fork graph with n sinks. Consider an instance (G, μ, c, L, o, g, P), such that $\mu(y) = \mu$ for all sources y of G. There is a minimum-length schedule for (G, μ, c, L, o, g, P) in which the tasks of G are scheduled on at most $\min\{n, P\}$ processors. A minimum-length schedule for (G, μ, c, L, o, g, P) is constructed by computing the length of a minimum-length m-processor schedule for all positive integers $m \leq \min\{n, P\}$. These lengths are used to construct a minimum-length schedule for (G, μ, c, L, o, g, P).

Let G be a fork graph with n sinks. Consider an instance (G, μ, c, L, o, g, P), such that all sinks y of G have execution length $\mu(y) = \mu$. Let $m \leq \min\{n, P\}$ be a positive integer. In an m-processor schedule for (G, μ, c, L, o, g, P), $c(x)$ receive operations have to be executed on $m-1$ processors and at least one sink is scheduled after the last receive operation on each of these processors. Hence $C_m = (m-1)c(x)$ send and receive operations are executed in an m-processor schedule for (G, μ, c, L, o, g, P). Because the length of a minimum-length 1-processor schedule for (G, μ, c, L, o, g, P) equals $\mu(x) + n\mu$, we will only consider the computation of the length of minimum-length m-processor schedules for (G, μ, c, L, o, g, P), where $m \geq 2$.

First we will consider an m-processor schedule $(\sigma_{m,0}, \pi_{m,0})$ for (G, μ, c, L, o, g, P), in which the communication operations are executed as early as possible. We may assume that x is scheduled at time 0 on processor 1 and that send operations $s_{x,p,i}$ are executed before send operations $s_{x,p+1,j}$ for all processors $p \in \{2, \ldots, m-1\}$ and all $i, j \in \{1, \ldots, c(x)\}$. So we may assume that for all processors $p \in \{2, \ldots, m\}$ and all $i \in \{1, \ldots, c(x)\}$, send operation $s_{x,p,i}$ is scheduled at time $\mu(x) + ((p-2)c(x) + i - 1)\max\{o, g\}$ and receive operation $r_{x,p,i}$ at time $\mu(x) + ((p-2)c(x) + i - 1)\max\{o, g\} + L + o$. Hence the last send operation finishes at time

$$idle_{m,0}(1) \;=\; \mu(x) + ((m-1)c(x) - 1)\max\{o, g\} + o.$$

Since we may assume that the sinks of G are scheduled immediately after the last communication operation on processors $2, \ldots, m$, the first sink on processor $p \in \{2, \ldots, m\}$ finishes at time

$$idle_{m,0}(p) \;=\; \mu(x) + ((p-1)c(x) - 1)\max\{o, g\} + L + 2o + \mu.$$

Now consider a minimum-length m-processor schedule (σ_m, π_m) for (G, μ, c, L, o, g, P). We may assume that the communication operations are scheduled in the same order and on the same processors as in $(\sigma_{m,0}, \pi_{m,0})$. The sinks of G are scheduled after the receive operations on one of the processors $2, \ldots, m$ or after or between the send operations on processor 1.

There is a delay of at least $\max\{o, g\} - o$ time units between the completion time of a send operation and the starting time of the next one. Let $\alpha(o, g) = \frac{\max\{o,g\}-o}{\mu}$. If there is a delay of $\max\{o, g\}$ time units between the starting times of two consecutive send operations, then at most $\lfloor \alpha(o, g) \rfloor$ sinks can be scheduled between them. If at least $\lceil \alpha(o, g) \rceil$ sinks are scheduled between two consecutive send operations, then we may assume that processor 1 is not idle between these send operations. It is not difficult to see that if more than $\lceil \alpha(o, g) \rceil$ sinks are scheduled between two consecutive send operations, then one of them can be scheduled at a later time without increasing the schedule length. Hence we may assume that at most $\lceil \alpha(o, g) \rceil$ sinks are scheduled between two consecutive send operations. In addition, we may assume that no sink is scheduled before the first send operation on processor 1. So the total number of sinks that are scheduled between the send operations on processor 1 is at most $(C_m - 1)\lceil \alpha(o, g) \rceil$.

If $\lceil \alpha(o, g) \rceil$ sinks are scheduled between two consecutive send operations s_1 and s_2, then the starting times of these send operations differs exactly $o + \lceil \alpha(o, g) \rceil \mu$. So compared to the

starting times of s_1 and s_2 in $(\sigma_{m,0}, \pi_{m,0})$, the starting time of s_2 is increased by

$$inc(o,g) \;=\; \lceil \alpha(o,g) \rceil \mu - (\max\{o,g\} - o).$$

Assume k sinks are scheduled between the send operations on processor 1. We may assume that $k \leq (C_m - 1)\lceil \alpha(o,g) \rceil$ and $k \leq n - m + 1$. In addition, because $\lfloor \alpha(o,g) \rfloor$ sinks can be scheduled between any pair of consecutive send operations without increasing the schedule length, we may assume that at least $\min\{n - m + 1, (C_m - 1)\lfloor \alpha(o,g) \rfloor\}$ sinks are scheduled between the send operations on processor 1. If $k = k_0 + (C_m - 1)\lfloor \alpha(o,g) \rfloor$ for some non-negative integer k_0, then $\lceil \alpha(o,g) \rceil$ sinks have to be scheduled before the last k_0 send operations and $\lfloor \alpha(o,g) \rfloor$ before the other send operations except the first. If $k \leq (C_m - 1)\lfloor \alpha(o,g) \rfloor$, then at most $\lfloor \alpha(o,g) \rfloor$ sinks have to be scheduled between any pair of consecutive send operations on processor 1. Hence the last send operation on processor 1 finishes

$$inc_{m,k}(1) \;=\; \max\{0, k - (C_m - 1)\lfloor \alpha(o,g) \rfloor\}\, inc(o,g)$$

time units later than in $(\sigma_{m,0}, \pi_{m,0})$. Moreover, the completion times of the first sinks on processors $2, \ldots, m$ are increased compared to their completion times in $(\sigma_{m,0}, \pi_{m,0})$. The send operations $s_{x,p,i}$ are scheduled before send operations $s_{x,p+1,j}$ for all processors $p \in \{2, \ldots, m-1\}$ and all $i, j \in \{1, \ldots, c(x)\}$. Because $\lceil \alpha(o,g) \rceil$ sinks are scheduled between the last k_0 pairs of consecutive send operations on processor 1, the completion times of the first sink on the last $\lceil \frac{k_0}{c(x)} \rceil$ processors are increased. The completion time of the first sink on processor $p \in \{2, \ldots, m\}$ is increased by

$$inc_{m,k}(p) \;=\; \max\{0, k - (C_m - 1)\lfloor \alpha(o,g) \rfloor - (m-p)c(x)\}\, inc(o,g),$$

because $\lceil \alpha(o,g) \rceil$ sinks are scheduled before the last $k_0 = k - (C_m - 1)\lfloor \alpha(o,g) \rfloor$ send operations on processor 1 and the $(m-p)c(x)$ send operations scheduled on processor 1 after send operation $s_{x,p,c(x)}$ does not increase the starting time of the first sink on processor p.

Let $\ell_{m,k}$ be the minimum length of an m-processor schedule for (G, μ, c, L, o, g, P) in which k sinks are scheduled between the send operations on processor 1. Then $\ell_{m,k}$ is the length of (σ_m, π_m). So we may assume that the last send operation on processor 1 finishes at time

$$idle_{m,k}(1) \;=\; idle_{m,0}(1) + inc_{m,k}(1)$$

and that for all processors $p \in \{2, \ldots, m\}$, the completion time of the first sink on processor p equals

$$idle_{m,k}(p) \;=\; idle_{m,0}(p) + inc_{m,k}(p).$$

Note that $idle_{m,k}(m) \geq idle_{m,k}(p)$ for all processors $p \in \{1, \ldots, m\}$. Since the remaining $n - k$ sinks have to be scheduled after the send operations on processor 1 or after the first sink on a processor $p \in \{2, \ldots, m\}$, $\ell_{m,k}$ is the smallest integer ℓ, such that

$$\ell \;\geq\; idle_{m,k}(m) \qquad \text{and} \qquad \sum_{p=1}^{m} \left\lfloor \frac{\ell - idle_{m,k}(p)}{\mu} \right\rfloor \;\geq\; n - k.$$

Define

$$\ell_{m,k,0} \;=\; \min\{\ell \in \mathbb{Q} \mid \ell \geq idle_{m,k}(m) \wedge \sum_{p=1}^{m} \frac{\ell - idle_{m,k}(p)}{\mu} \geq n - k\}.$$

Then $\ell_{m,k,0} \leq \ell_{m,k} < \ell_{m,k,0} + \mu$. $\ell_{m,k,0}$ can be computed in $O(m)$ time:

$$\ell_{m,k,0} \;=\; \max\{idle_{m,k}(m), \frac{1}{m}((n-k)\mu + \sum_{p=1}^{m} idle_{m,k}(p))\}.$$

If $\ell_{m,k,0} = idle_{m,k}(m)$, then $\ell_{m,k,0} = \ell_{m,k} = idle_{m,k}(m)$. So we will assume that $\ell_{m,k,0} \neq idle_{m,k}(m)$. Then

$$\ell_{m,k} \;=\; \min\{\ell \in \mathbb{Z} \mid \sum_{p=1}^{m} \left\lfloor \frac{\ell - idle_{m,k}(p)}{\mu} \right\rfloor = \sum_{p=1}^{m} \frac{\ell_{m,k,0} - idle_{m,k}(p)}{\mu}\}.$$

Since $\ell_{m,k,0} \neq idle_{m,k}(m)$, $\sum_{p=1}^{m} \frac{\ell_{m,k,0} - idle_{m,k}(p)}{\mu} \in \mathbb{N}$. Define

$$D \;=\; \sum_{p=1}^{m} \frac{\ell_{m,k,0} - idle_{m,k}(p)}{\mu} - \sum_{p=1}^{m} \left\lfloor \frac{\ell_{m,k,0} - idle_{m,k}(p)}{\mu} \right\rfloor.$$

Note that $D \in \mathbb{N}$ and $D \leq m$. Assume that for all processors $p \in \{1, \ldots, m\}$,

$$\ell_{m,k,0} - idle_{m,k}(p) \;=\; q_p\mu + r_p,$$

such that $0 \leq r_p < \mu$. Then $\ell_{m,k} - \ell_{m,k,0}$ equals the smallest $d \in \mathbb{Q}$, such that $\ell_{m,k,0} + d \in \mathbb{Z}$ and for at least D processors p, $r_p + d \geq \mu$. Then $\ell_{m,k}$ can be computed as follows. Select the D^{th} element in the list of processors ordered by non-increasing r_p-values. Assume the D^{th} processor in this list is processor p_0. Then

$$\ell_{m,k} \;=\; \lceil \ell_{m,k,0} + \mu - r_{p_0} \rceil.$$

Selecting the D^{th} processor takes $O(m)$ time [7], so $\ell_{m,k}$ can be computed in $O(m)$ time.

Let $\ell^*_m = \min_k \ell_{m,k}$ and $\ell^* = \min_{1 \leq m \leq \min\{n,P\}} \ell^*_m$. Then ℓ^*_m is the length of a minimum-length m-processor schedule for (G, μ, c, L, o, g, P) and ℓ^* the length of a minimum-length schedule for (G, μ, c, L, o, g, P). For each positive integer $m \leq \min\{n, P\}$, ℓ^*_m can be computed in $O(n^2)$ time, because $c(x)$ is bounded by a constant. So ℓ^* can computed in $O(n^3)$ time. If ℓ^* equals $\ell_{m,k}$, then m and k can be used to construct a minimum-length schedule in linear time. Hence we have proved the following result.

Theorem 5.1. *There is an algorithm with an $O(n^3)$ time complexity that constructs minimum-length schedules for instances (G, μ, c, L, o, g, P), such that G is a fork graph with n sinks and there is a positive integer μ, such that $\mu(y) = \mu$ for all sinks y of G.*

If $\max\{o, g\} - o$ is divisible by μ (for instance, if $g \leq o$ or if $\mu = 1$), then the length of a minimum-length schedule for (G, μ, c, L, o, g, P) can be computed more efficiently. Assume $\max\{o, g\} - o$ is divisible by μ. Then $\alpha(o, g) \in \mathbb{N}$. So we may assume that in a minimum-length m-processor schedule for (G, μ, c, L, o, g, P), exactly $k_m = \min\{n, (C_m - 1)\alpha(o, g)\}$ sinks of G are scheduled between the send operations on processor 1. Obviously, $inc_{m,k_m}(p) = 0$ for all processors $p \in \{1, \ldots, m\}$. So in a minimum-length m-processor schedule for (G, μ, c, L, o, g, P), the last send operation on processor 1 finishes at time

$$idle_{m,k_m}(1) \;=\; idle_{m,0}(1) \;=\; \mu(x) + ((m-1)c(x) - 1)\max\{o, g\} + o.$$

The completion time of the first sink on processor $p \in \{2, \ldots, m\}$ equals

$$idle_{m,k_m}(p) \;=\; idle_{m,0}(p) \;=\; \mu(x) + ((p-1)c(x) - 1)\max\{o, g\} + L + 2o + \mu.$$

Moreover, ℓ^*_m is the smallest integer ℓ, such that

$$\ell \;\geq\; idle_{m,k_m}(m) \qquad \text{and} \qquad \sum_{p=1}^{m} \left\lfloor \frac{\ell - idle_{m,k_m}(p)}{\mu} \right\rfloor \;\geq\; n - k_m.$$

ℓ^*_m can be computed in $O(n)$ time. Hence $\ell^* = \min_{1 \leq m \leq \min\{n,P\}} \ell^*_m$ can be computed in $O(n^2)$ time. Given the number of processors m, such that $\ell^* = \ell^*_m$, a minimum-length schedule for (G, μ, c, L, o, g, P) can be constructed in linear time. So we have proved the following result.

Theorem 5.2. *There is an algorithm with an $O(n^2)$ time complexity that constructs minimum-length schedules for instances (G, μ, c, L, o, g, P), such that G is a fork graph with n sinks and there is a positive integer μ, such that $\mu(y) = \mu$ for all sinks y of G and $\max\{o, g\} - o$ is divisible by μ.*

6 Concluding remarks

In this paper, we studied the complexity of scheduling fork graphs in the LogP model. It was shown that constructing minimum-length schedules for fork graphs is a strongly NP-hard optimisation problem. In addition, two polynomial-time algorithms were presented that construct schedules for fork graphs; one is a 2-approximation algorithm for scheduling arbitrary fork graphs, the other constructs minimum-length schedules for fork graphs in which all sinks have the same execution length. The basis of these algorithms is the knowledge of the structure of a minimum-length m-processor schedule for a fork graph.

A similar approach can be used for scheduling join graphs in the LogP model. The structure of a minimum-length schedule for a join graph can be used to construct schedules for join graphs. If g does not exceed o, then a schedule on an unrestricted number of processors that is at most three times as long as a minimum-length schedule can be constructed in polynomial time and for each constant $k \geq 1$, a schedule on a restricted number of processors that is at most $3 + \frac{1}{k}$ times as long as minimum-length schedules can be constructed in polynomial time [24]. In addition, if all tasks of a join graph have the same execution length, then a minimum-length schedule on an unrestricted number of processors can be constructed in polynomial time [24].

References

[1] M. Adler, J.W. Byers and R.M. Karp. Parallel sorting with limited bandwidth. In *Proceedings of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 129–136, 1995.

[2] A. Aggarwal, A.K. Chandra and M. Snir. On communication latency in PRAM computations. In *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pages 11–21, 1989.

[3] A. Aggarwal, A.K. Chandra and M. Snir. Communication complexity of PRAMs. *Theoretical Computer Science*, 71(1):3–28, March 1990.

[4] A. Bar-Noy and S. Kipnis. Designing broadcasting algorithms in the Postal model for message-passing systems. *Mathematical Systems Theory*, 27(5):431–452, September/October 1994.

[5] P. Chrétienne and C. Picouleau. Scheduling with communication delays: a survey. In P. Chrétienne, E.G. Coffman, Jr., J.K. Lenstra and Z. Liu, editors, *Scheduling Theory and its Applications*, Chapter 4, pages 65–90. John Wiley & Sons, Chichester, United Kingdom, 1995.

[6] R. Cole and O. Zajicek. The APRAM: Incorporating asynchrony into the PRAM model. In *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pages 169–178, 1989.

[7] T.H. Cormen, C.E. Leiserson and R.L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge MA, United States, 1990.

[8] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of the Fourth ACM-SIGPLAN Symposium on Principles and Practice of Parallel Processing*, pages 1–12, 1993.

[9] D.E. Culler, R.M. Karp, D. Patterson, A. Sahay, E.E. Santos, K.E. Schauser, R. Subramonian and T. von Eicken. LogP: A practical model of parallel computation. *Communications of the ACM*, 39(11):78–85, November 1996.

[10] A.C. Dusseau, D.E. Culler, K.E. Schauser and R.P. Martin. Fast parallel sorting under LogP: Experience with the CM-5. *IEEE Transactions on Parallel and Distributed Systems*, 7(8):791–805, August 1996.

[11] S. Fortune and J. Wyllie. Parallelism in Random Access Machines. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, pages 114–118, 1978.

[12] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York NY, United States, 1979.

[13] P.B. Gibbons. A more practical PRAM model. In *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pages 158–168, 1989.

[14] P.B. Gibbons, Y. Matias and V. Ramachandran. Efficient low-contention parallel algorithms. In *Proceedings of the 6th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 236–247, 1994.

[15] P.B. Gibbons, Y. Matias and V. Ramachandran. The QRQW PRAM: Accounting for contention in parallel algorithms. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 638–648, 1994.

[16] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[17] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, March 1969.

[18] R.M. Karp, A. Sahay, E.E. Santos and K.E. Schauser. Optimal broadcast and summation in the LogP model. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 142–153, 1993.

[19] I. Kort and D. Trystram. Some results on scheduling trees of height one under logp. Unpublished manuscript, 1997.

[20] W. Löwe and W. Zimmermann. Upper time bounds for executing PRAM-programs on the LogP-machine. In *Proceedings of the 9th ACM International Conference on Supercomputing*, pages 41–50, 1995.

[21] W. Löwe, W. Zimmermann and J. Eisenbiegler. On linear schedules for task graphs for

generalized LogP-machines. In C. Lengauer, M. Griebl and S. Gorlatch, editors, *Proceedings of the Third Euro-Par Conference*, Lecture Notes in Computer Science, volume 1300, pages 895–904. Springer-Verlag, Berlin, Germany, 1997.

[22] C. Martel and A. Raghunathan. Asynchronous PRAMs with memory latency. *Journal of Parallel and Distributed Computing*, 23(1):10–26, October 1994.

[23] L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.

[24] J. Verriet. *Scheduling with communication for multiprocessor computation.* PhD thesis, Utrecht University, Utrecht, the Netherlands, June 1998.

[25] W. Zimmermann and W. Löwe. An approach to machine-independent parallel programming. In B. Buchberger and J. Volkert, editors, *Proceedings of the Third Joint Conference on Vector and Parallel Processing*, Lecture Notes in Computer Science, volume 854, pages 277–288. Springer-Verlag, Berlin, Germany, 1994.