Scheduling tree-structured programs 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.ruu.nl

Abstract

The LogP model is a model of parallel computation that characterises a parallel computer architecture by four parameters: the latency L, the overhead o, the gap g and the number of processors P. We study the problem of constructing minimum-length schedules for tree-structured programs in the LogP model. This problem is proved to be NP-hard, even for outtrees of height two in LogP models with an unlimited number of processors.

For outtrees of height two, a 2-approximation algorithm is presented. For intrees of height two, two approximation algorithms are presented: a 3-approximation algorithm for LogP models with an unrestricted number of processors and a $4 - \frac{2}{P}$-approximation algorithm for LogP models with a finite number of processors.

For the problem of constructing minimum-length schedules for d-ary intrees in a LogP model with a finite number of processors, three approximation algorithms are presented that are applicable in many models of parallel computation. The first constructs schedules for full d-ary intrees of length at most $2 + \frac{2}{d}$ times the length of an optimal schedule plus the time required for $(d+1)P - 1$ communication operations. The second constructs schedules on P processors of length at most $d + 1 - \frac{d^2+d}{d+P}$ times the length of a minimum-length schedule plus the time needed for $d(P-1) - 1$ communication operations. The third constructs schedules of length at most $3 - \frac{6}{P+2}$ times the length of a minimum-length schedule plus the duration of $d(d-1)(P-1) - 1$ communication operations.

1 Introduction

The PRAM [10] is the most common model of parallel computation. A PRAM consists of a collection of processors that execute a parallel 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 parallel computer programs: a communication step takes the same amount of time as a local computation step, whereas, in a real parallel computer architecture, a communication step is far more time consuming. There are several PRAM-based models that include aspects of real parallel machines, such as latency [2, 3, 21], memory contention [14, 13] and asynchrony [6, 12].

The BSP model [22] and the LogP model [7, 9] are models of parallel computation that consist of a collection of processors that communicate using message passing. These models are more realistic, because they include several aspects of real parallel computers. Both models characterise a parallel architecture by a few parameters that capture memory latency and bandwidth. In this report, we will consider the LogP model that provides more control over the machine resources than the BSP model.

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

The LogP model consists of a number of processors connected by a communication network. Each processor has an unlimited amount of local memory. The processors process a parallel computer program in an asynchronous manner. Communication is modelled by message-passing: messages are sent from one processor to another via the communication network.

The LogP model characterises a parallel computer architecture by four parameters: the latency L, the overhead o, the gap g and the number of processors P. The latency is an upper bound on the time required to send a message from one processor to another via the communication network. The latency depends on the diameter of the network topology. The overhead is the amount of time a processor is involved in sending or receiving a message consisting of one word. During this time, a processor cannot perform other operations. The gap is the minimum delay between two consecutive message transmissions or receptions on the same processor.

Like for many other models of parallel computation, little is known about scheduling parallel programs in the LogP model. A few algorithms have been presented that construct schedules for common parallel programs in the LogP model. These programs include Fast Fourier Transform [7], sorting [1, 8] and broadcast [17].

In addition, Löwe and Zimmermann [20, 23] presented an algorithm that constructs schedules for communication structures G of PRAMs in LogP models with an unrestricted number of processors. The length of these schedules is at most $1 + \frac{1}{g(G)}$ times the length of an optimal schedule, where $g(G)$ is the grain size of G. Moreover, they presented an algorithm that constructs schedules of length at most twice as long as optimal plus the duration of the number of sequential communication operations.

In this paper, we will consider the problem of scheduling arbitrary parallel programs in the LogP model. We will restrict ourselves to programs that have a tree-like structure. Such programs include divide-and-conquer algorithms, such as the evaluation of arithmetic expressions.

In Section 3, we consider send graphs (outtrees of height two). It will be shown that, for all fixed parameters (L, o, g, P), such that $\max\{o, g\} \geq 1$ and $P \in \{2, 3, \ldots, \infty\}$, constructing minimum-length schedules for send graphs in the LogP model is an NP-hard problem. In this section, we also present a 2-approximation algorithm for scheduling send graphs. For the special case that $g \leq o$, the approximation bound of this algorithm equals $2 - \frac{1}{P}$.

Section 4 studies the problem of scheduling receive graphs (intrees of height two). For the case g does not exceed o, two approximation algorithms are presented. The first constructs schedules of length at most three times the length of an optimal schedule if the number of processors is unrestricted. The other algorithm is a $4 - \frac{2}{P}$-approximation algorithm for the case that the number of processors is bounded.

In Section 5, an approximation algorithm for scheduling d-ary intrees is presented. This algorithm uses a decomposition of a d-ary intree to construct a schedule in which the number of communication operations is small. In addition, three algorithms are presented that construct decompositions of d-ary intrees. The first constructs decompositions of full d-ary intrees with unit-length tasks. Using these decompositions, the approximation algorithm constructs schedules of length at most $2 + \frac{2}{d}$ times the length of a minimum-length schedule plus the time required for at most $(d+1)P - 1$ communication operations. For full binary intrees, the schedules are of length at most twice the length of an optimal schedule plus the duration of $2P - 1$ communication operations.

The second and third decomposition algorithm construct decompositions of arbitrary d-ary intrees with arbitrary task lengths. Using the decompositions constructed by the second decomposition algorithm, the approximation algorithm constructs schedules whose lengths are at most $d + 1 - \frac{d^2+d}{d+P}$ times the length of a minimum-length schedule plus the time needed for at most $d(P-1) - 1$ communication operations. Using the decompositions of the third decomposition algorithm, the approximation algorithm constructs schedules whose lengths are at most the sum of $3 - \frac{6}{P+2}$ times the length of an optimal schedule and the duration of $d(d-1)(P-1) - 1$

communication operations.

2 Preliminary definitions

In this paper, we study the problem of scheduling parallel programs in the LogP model. Such a program is represented by a triple (G, μ, c), such that $G = (V, E)$ is a directed acyclic graph (or precedence graphs), $\mu : V \to \mathbb{Z}^+$ and $c : V \to \mathbb{N}$. An element of V will be called a task (or node) of G. A task u corresponds to a task of the parallel program. u has a length $\mu(u)$: the execution of u on a processor takes $\mu(u)$ time. The number of messages required to store the result of the execution of u equals $c(u)$. If the result of the execution of u has to be sent to another processor, then $c(u)$ messages have to be sent through the communication network to the other processor. An instance of a LogP scheduling problem is represented by a tuple (G, μ, c, L, o, g, P), where (G, μ, c) represents the parallel program and L, o, g and P are 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 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 $u_1 \prec u_2$. By $u_1 \prec_0 u_2$, we denote that u_2 is an *immediate successor* of u_1 and u_1 an *immediate predecessor* of u_2. In that case, u_2 is a successor of u_1 and there is no task v such that $u_1 \prec v$ and $v \prec u_2$.

The *outdegree* of a task u is the number of immediate successors of u. Similarly, the number of immediate predecessors of u is the *indegree* of u. A task with indegree zero will be called a *source*, a task without successors is a *sink*.

In this paper, we consider special types of precedence graphs, inforests and outforests. An *inforest* is a graph in which every task has outdegree at most one. An *outforest* is an inforest in which the arcs have been reversed. Hence in an outforest, every task has at most one immediate predecessor. An *intree* is an inforest with exactly one task without successors. Similarly, an *outtree* is an outforest with exactly one task without predecessors.

Consider an inforest T. A task of T with outdegree zero is called a *root*; a *leaf* is a task without predecessors. The immediate predecessors of a task are called its *children* and its immediate successor is called its *parent*. $T(u)$ denotes the subtree of T induced by u and its predecessors. T is a *d-ary intree* if every task of T has indegree at most d. Since every task in an intree has at most one immediate successor, there is a unique path from a task to a successor. If u_1 is a predecessor of u_2, then $p(u_1, u_2)$ is the set containing the tasks on the path from the parent of u_1 to u_2.

For outforests, the same terms are used but with a different meaning. A *leaf* is a task without successors, a *root* has no predecessors. The *parent* of a task is its immediate predecessor, its *children* are its immediate successors. Similar to inforests, a *d-ary outforest* is an outforest in which every task has at most d children. For outforests, $T(u)$ denotes the subtree consisting of task u and its successors.

Consider an instance (G, μ, c, L, o, g, P). Let u_1 and u_2 be two tasks of G and suppose u_2 is an immediate successor of u_1. Then the result of u_1 is needed to execute u_2. If u_1 and u_2 are executed on different processors, then $c(u_1)$ messages must be sent from one processor to another. A message can be received L time after it has been sent. After all messages have been received, u_2 can be executed. Sending and receiving a message takes o time and the delay between sending or receiving two messages is of length at least g. Figure 1 shows the execution of u_1 and u_2 on different processors in case $c(u_1)$ equals three. The tasks s_i and r_i represent the transmission and reception of the i^{th} message.

For each task u, send and receive tasks have to be scheduled for each processor on which an immediate successor of u is scheduled. No message has to be sent twice to the same processor. Hence we will define two sets S and R containing the send and the receive tasks. S contains tasks $s(u, p, i)$, such that u is a task of G, p is a processor and $i \leq c(u)$ is the number of the message

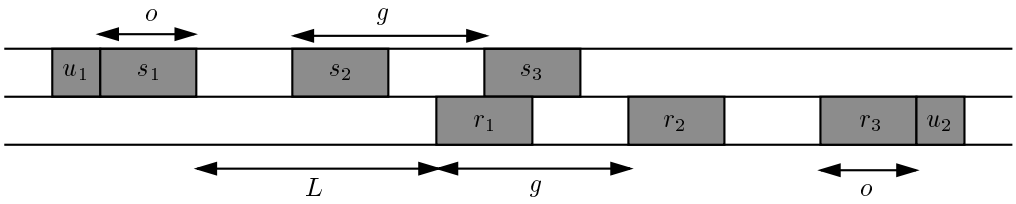

Figure 1: Communication between two processors

of u. R contains elements $r(u,p,i)$. The tasks $s(u,p,i)$ and $r(u,p,i)$ represent the transmission and reception of the i^{th} message of u in case of transmission of the result of the execution of u to processor p. Let U be the union of the set of tasks of G, the set of send tasks S and the set of receive tasks R. A task u of S or R has length $\mu(u) = o$.

A *schedule* for (G, μ, c, L, o, g, P) is a pair of functions (σ, π), such that $\sigma : U \to \mathbb{N} \cup \{\perp\}$ and $\pi : U \to \{1, \ldots, P, \perp\}$. σ assigns a starting time to every task of U and π a processor. The value $\perp$ denotes the starting time and processor of (communication) tasks that are not scheduled. A schedule (σ, π) is *feasible* 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 U, 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 $u_1 \prec_0 u_2$, then $\sigma(u_1) + \mu(u_1) \leq \sigma(u_2)$;
4. for all tasks u_1 and u_2 of G, if $u_1 \prec_0 u_2$ 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 elements u_1 and u_2 of S or R, if $\pi(u_1) = \pi(u_2) \neq \perp$, then $\sigma(u_1) + g \leq \sigma(u_2)$ or $\sigma(u_2) + g \leq \sigma(u_1)$; and
6. for all tasks u of G and all processors p, if no immediate successors of u are scheduled on processor p or $p = \pi(u)$, then $\sigma(s(u,p,i)) = \perp$ and $\pi(r(u,p,i)) = \perp$.

The *completion time* of a task u in a schedule (σ, π) equals $\sigma(u) + \mu(u)$. The *length* of a schedule (σ, π) is the maximum completion time of a task. A schedule is called *optimal* if its length equals that of a minimum-length schedule.

In Figures 2 and 3, an example of a feasible schedule is presented. Figure 2 shows a precedence graph G. Assume $\mu(x) = 1$ and $c(x) = 3$. Let $\mu(y_1) = 1$, $\mu(y_2) = 1$, $\mu(y_3) = 2$, $\mu(y_4) = 3$ and $\mu(y_5) = 7$. In addition, let $L = 1$, $o = 1$, $g = 2$ and $P = 2$. Then the schedule shown in Figure 3 is a feasible schedule for (G, μ, c, L, o, g, P) of length 14. Task s_i corresponds to send task $s(x, 2, i)$ and task r_i to receive task $r(x, 2, i)$. This is an optimal schedule for (G, μ, c, L, o, g, P).

Note that tasks y_1 and y_2 are scheduled between the send tasks. No task can be executed between the receive tasks, because all three messages are needed to send the result of the execution of x to another processor. Although two successors of x are executed on processor 2, only three send and receive tasks are executed. This is due to the fact that we consider communication between processors, not between processes (or tasks): the result of the execution of x is sent to processor 2, not to tasks y_3 and y_4.

3 Send graphs

In this section, we study the problem of scheduling send graphs in the LogP model. A send graph is an outtree of height two. Hence a send graph consists of a source x and its children $y_1, \ldots, y_n$.

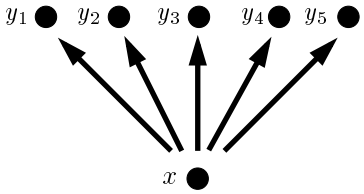

Figure 2: A precedence graph G

Figure 3: A feasible schedule for (G, μ, c, L, o, g, P)

These children are the sinks of the graph. Figure 4 shows a send graph.

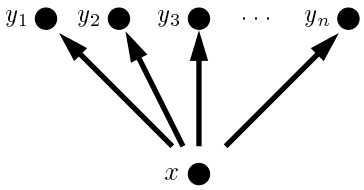

Figure 4: A send graph

We will show that constructing minimum-length schedules for instances (G, μ, c, L, o, g, P), such that G is a send graph is NP-hard for all fixed choices of L, o, g and P, such that $\max\{o, g\} \geq 1$. Thereafter, a 2-approximation algorithm is presented. The approximation bound of this algorithm improves to $2 - \frac{1}{P}$ if $g \leq o$.

3.1 NP-completeness

In this section, we will show that constructing minimum-length schedules for send graphs is NP-hard. This is obvious if the number of processors is finite. Using a polynomial reduction from Partition, we will show that is is also true for all fixed choices of L, o, g and P, such that $\max\{o, g\} \geq 1$ and $P = \infty$. Partition [11] is defined as follows.

Problem. Partition
Instance. A set $A = \{a_1, \ldots, a_n\}$ of positive integers.
Question. Is there a subset A' of A, such that $\sum_{a \in A'} a = \frac{1}{2} \sum_{a \in A} a$?

Let L, o and g be non-negative integers and $P \in \{2, 3, \ldots, \infty\}$. (L, o, g, P)-LogP Send graph scheduling is the following problem.

Problem. (L, o, g, P)-LogP Send graph scheduling
Instance. A parallel program (G, μ, c), such that G is a send graph and an integer B.
Question. Is there a feasible schedule for (G, μ, c, L, o, g, P) of length at most B?

The following lemma presents a polynomial reduction from Partition to (L, o, g, P)-LogP Send graph scheduling.

Lemma 3.1. *For all non-negative integers L, o and g and all $P \in \{2, 3, \ldots, \infty\}$, such that $\max\{o, g\} \geq 1$, there is a polynomial reduction from* PARTITION *to* (L, o, g, P)-LOGP SEND GRAPH SCHEDULING.

Proof. Let L, o and g be non-negative integers and $P \in \{2, 3, \ldots, \infty\}$, such that $\max\{o, g\} \geq 1$. Consider an instance $A = \{a_1, \ldots, a_n\}$ of PARTITION. Let $N = \sum_{i=1}^{n} a_i$. Define $M = 2(L + o + 1)\max\{o, g\}N$ and $c = (L + o + 1)(N - 2) + 1$. Construct an instance (G, μ, c) of (L, o, g, P)-LOGP SEND GRAPH SCHEDULING as follows. G is a send graph with source x and sinks $y_1, \ldots, y_n$, such that $\mu(x) = 1$, $\mu(y_i) = 2(L + o + 1)\max\{o, g\}a_i$ and $c(x) = c$. In addition, let $B = 1 + (c - 1)\max\{o, g\} + 2o + L + \frac{1}{2}M$.

($\Rightarrow$) Assume A_1 is a subset of A, such that $\sum_{a \in A_1} a = \frac{1}{2}N$. Define $Y_1 = \{y_i \mid a_i \in A_1\}$ and $Y_2 = \{y_i \mid a_i \notin A_1\}$. Construct a schedule (σ, π) for (G, μ, c, L, o, g, P) as follows. x is scheduled at time 0 on processor 1. For $i \leq c(x)$, send task $s(x, 2, i)$ is executed at time $1+(i-1)\max\{o, g\}$ on processor 1 and receive task $r(x, 2, i)$ at time $1+(i-1)\max\{o, g\}+o+L$ on processor 2. The tasks of Y_1 are scheduled without interruption on processor 1 from time $1+(c-1)\max\{o, g\}+o$ until time $1+(c-1)\max\{o, g\}+o+\frac{1}{2}M$ and the tasks of Y_2 on processor 2 from time $1+(c-1)\max\{o, g\}+2o+L$ until time $1+(c-1)\max\{o, g\}+2o+L+\frac{1}{2}M = B$. Then no processor executes two tasks at the same time and the tasks on processor 2 are scheduled after the receive tasks. So (σ, π) is a feasible schedule for (G, μ, c, L, o, g, P) of length B.

($\Leftarrow$) Assume (σ, π) is a schedule for (G, μ, c, L, o, g, P) of length $\ell \leq B$. Suppose all tasks are scheduled on one processor. Then the length of (σ, π) is at least $M + 1$. However,

$$\begin{aligned}
\ell &\leq 1 + (c - 1)\max\{o, g\} + 2o + L + \frac{1}{2}M \\
&= 1 + (L + o + 1)\max\{o, g\}(N - 2) + \max\{o, g\} + 2o + L + \frac{1}{2}M \\
&= 1 + \frac{1}{2}M - 2(L + o + 1)\max\{o, g\} + \max\{o, g\} + 2o + L + \frac{1}{2}M \\
&\leq M + 1 - 2L - 2o - 2\max\{o, g\} + \max\{o, g\} + 2o + L \\
&< M + 1.
\end{aligned}$$

Hence the sinks of G are executed on at least two processors. If the tasks y_i are executed on at least three processors, then the first task on the third processor is executed after at least $2c$ receive tasks. Hence it does not start before time $1 + (2c - 1)\max\{o, g\} + 2o + L$. Since every sink has length at least $2(L + o + 1)\max\{o, g\}$,

$$\begin{aligned}
\ell &\geq 1 + (2c - 1)\max\{o, g\} + 2o + L + 2(L + o + 1)\max\{o, g\} \\
&= 1 + (c - 1)\max\{o, g\} + 2o + L + c\max\{o, g\} + 2(L + o + 1)\max\{o, g\} \\
&= 1 + (c - 1)\max\{o, g\} + 2o + L + \\
&\qquad (L + o + 1)(N - 2)\max\{o, g\} + \max\{o, g\} + 2(L + o + 1)\max\{o, g\} \\
&= 1 + (c - 1)\max\{o, g\} + 2o + L + (L + o + 1)\max\{o, g\}N + \max\{o, g\} \\
&= 1 + (c - 1)\max\{o, g\} + 2o + L + \frac{1}{2}M + \max\{o, g\} \\
&= B + \max\{o, g\}.
\end{aligned}$$

So the sinks of G are executed on exactly two processors. Let these be processors 1 and 2. We may assume that x is executed on processor 1. Define $A_1 = \{a_i \mid \pi(y_i) = 1\}$ and $A_2 = \{a_i \mid \pi(y_i) = 2\}$. Let $N_1 = \sum_{a \in A_1} a$ and $N_2 = \sum_{a \in A_2} a$. Since the length of (σ, π) is at most B and the first task on processor 2 cannot start before time $1+(c-1)\max\{o, g\}+2o+L$, the sum of the elements of A_2 does not exceed $\frac{1}{2}N$. It is not difficult to see that no sink y_i is scheduled before a send task on processor 1. If $N_1 > \frac{1}{2}N$, then the last task on processor 1 is not completed before time $1 + (c - 1)\max\{o, g\} + o + \frac{1}{2}M + 2(L + o + 1)\max\{o, g\} > B$. Contradiction. So $N_1 = \frac{1}{2}N$.

□

From Lemma 3.1, we can conclude that (L, o, g, P)-LogP Send graph scheduling is NP-complete for all fixed L, o, g and P, such that $\max\{o, g\} \geq 1$. Moreover, it is not difficult to prove that (L, o, g, P)-LogP Send graph scheduling is NP-complete for all L, o, g and P, such that $P \neq \infty$. Hence we obtain the following result.

Theorem 3.2. *For all fixed parameters (L, o, g, P), such that $\max\{o, g\} \geq 1$ or $P \neq \infty$, constructing minimum-length schedules for instances (G, μ, c, L, o, g, P), such that G is a send graph is* NP*-hard.*

If g and o equal zero, then scheduling in the LogP model is a special case of scheduling with arc-dependent communication delays (latencies). In that case, minimum-length schedules for send graphs on an unlimited number of processors can be constructed in polynomial time [5] and constructing minimum-length schedules for outtrees of height three is NP-hard, even if the number of children of the root equals the number of sinks [4].

3.2 A 2-approximation algorithm

In this section, we will present a simple 2-approximation algorithm for scheduling send graphs in the LogP model. It is obvious that the number of processors used in an optimal schedule for a send graph instance does not exceed the number of sinks or the number of processors P. For each possible number of processors m, the algorithm constructs a schedule that is at most twice as long as a schedule that uses m processors of minimum length. By choosing the shortest schedule, a schedule is constructed whose length is at most twice the length of an optimal schedule.

Consider a LogP instance (G, μ, c, L, o, g, P), such that G is a send 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) that uses at most $\min\{n, P\}$ processors. Let $m \leq \min\{n, P\}$. Define ℓ_m^* as the length of a minimum-length m-processor schedule, where an *m-processor schedule* for (G, μ, c, L, o, g, P) is a schedule for (G, μ, c, L, o, g, P), in which the sinks of G are executed on exactly m processors. A minimum-length m-processor schedule will be called *m-processor optimal*. Let ℓ^* be the length of an optimal schedule for (G, μ, c, L, o, g, P). Then $\ell^* = \min_{m \leq \min\{n,P\}} \ell_m^*$.

In an m-processor schedule for (G, μ, c, L, o, g, P), $c(x)$ receive tasks have to be executed on $m-1$ processors. The first step of the algorithm constructs a communication profile in which the send and receive tasks are scheduled as early as possible. x starts at time 0 on processor 1. The send tasks $s(x, p, i)$ are scheduled on processor 1 at times $\mu(x) + ((p-2)c(x) + i - 1)\max\{o, g\}$. The corresponding receive tasks $r(x, p, i)$ on processor p at times $\mu(x) + ((p-2)c(x) + i - 1)\max\{o, g\} + o + L$. Then the last send task is completed at time $idle(1) = \mu(x) + ((m-1)c(x) - 1)\max\{o, g\} + o$ and the last receive task on processor p at time $idle(p) = \mu(x) + ((p-1)c(x) - 1)\max\{o, g\} + 2o + L$. The sinks y_i with $i \leq m$ are scheduled immediately after the last receive task on processor i and $idle(i)$ is increased by $\mu(y_i)$.

The remaining sinks $y_{m+1}, \ldots, y_n$ are scheduled after the communication tasks and sinks $y_1, \ldots, y_m$ on one of the processors. This is done by a straightforward modification of Graham's List scheduling algorithm [15]. For each task y_i, choose a processor p, such that $idle(p)$ is minimal. Schedule y_i at time $idle(p)$ on processor p and increase $idle(p)$ by $\mu(y_i)$.

Let (σ_m, π_m) be the schedule constructed by these two steps. Let ℓ_m be the length of (σ_m, π_m). Assume y_i is a task that finishes at time ℓ_m. If y_i is the first sink scheduled on processor $\pi_m(y_i)$, then it is scheduled immediately after receive task $r(x, \pi_m(y_i), c(x))$. Every m-processor schedule has length at least $\mu(x) + ((m-1)c(x) - 1)\max\{o, g\} + 2o + L$, because $(m-1)c(x)$ receive tasks have to be executed. So the completion time of the last receive task on processor $\pi_m(y_i)$ is a lower bound of ℓ_m^*. So is the execution length of y_i. So $\ell_m = \sigma_m(y_i) + \mu(y_i) \leq 2\ell_m^*$. Therefore (σ_m, π_m) is at most twice as long as an m-processor optimal schedule.

Otherwise, suppose that another sink is scheduled before y_i. If a processor become idle before time $\sigma_m(y_i)$, then y_i would have been scheduled at an earlier time. So all processors are busy until time $\sigma_m(y_i)$. Every m-processor schedule has length at least $\mu(x) + ((m-1)c(x)-1)\max\{o,g\} + 2o + L$, because $c(x)$ receive tasks are executed on $m-1$ processors. On processors that do not execute x, no task can be executed before a receive task. Hence the idle periods in (σ_m, π_m) on processors $2, \ldots, m$ before the last receive task cannot be avoided. Hence the only idle time in (σ_m, π_m) that can be avoided is the idle time between the send tasks on processor 1. As a result,

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

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

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

Note that, if $g \leq o$, then $\ell_m \leq (2 - \frac{1}{m})\ell_m^*$.

For each $m \leq \min\{n, P\}$, construct an m-processor schedule (σ_m, π_m) of length ℓ_m. Assume (σ_k, π_k) is the shortest of these schedules and there is an optimal schedule in which sinks are executed on k^* processors. Then $\ell_k \leq \ell_{k^*} \leq 2\ell_{k^*}^* = 2\ell^*$. If $g \leq o$ and $P \neq \infty$, then $\ell_k \leq \ell_{k^*} \leq (2 - \frac{1}{k^*})\ell_{k^*}^* \leq (2 - \frac{1}{P})\ell_{k^*}^* = (2 - \frac{1}{P})\ell^*$.

Hence we have proved the following theorem.

Theorem 3.3. *Let (G, μ, c, L, o, g, P) be a LogP instance, such that G is a send graph. Then, in polynomial time, a schedule for (G, μ, c, L, o, g, P) can be constructed whose length is at most twice the length of an optimal schedule for (G, μ, c, L, o, g, P). If $g \leq o$, then this schedule has length at most $2 - \frac{1}{P}$ times that of an optimal schedule for (G, μ, c, L, o, g, P).*

4 Receive graphs

In this section, we will consider the problem of scheduling receive graphs in the LogP model. A receive graph is a send graph in which the arcs have been reversed. Hence a receive graph consists of a sink x with children $y_1, \ldots, y_n$, that are the sources of the graph. A receive graph is shown in Figure 5.

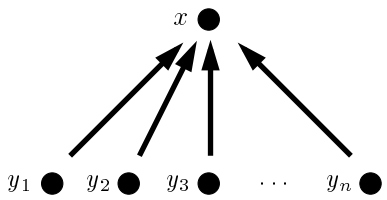

Figure 5: A receive graph

We will consider the case in which $g \leq o$. For this special case, two approximation algorithms are presented. The first approximation constructs schedules for receive graph instances in LogP

models with an unrestricted number of processors. The length of these schedules is at most three times the length of an optimal schedule. The other algorithm is a $4 - \frac{2}{P}$-approximation algorithm for scheduling in LogP models with a finite number of processors.

4.1 A 3-approximation algorithm for the unrestricted case

In this section, an approximation algorithm for scheduling receive graphs in LogP models with an unrestricted number of processors is presented. For this algorithm, we will assume that g does not exceed o. It is a 3-approximation algorithm for scheduling receive graphs in LogP models with an unrestricted number of processors. It corresponds to the 3-approximation algorithm of Hsu and Lopez [16] for scheduling send and receive graphs in a model of parallel computation that resembles the LogP model.

Consider an instance (G, μ, c, L, o, g, P), such that G is a receive graph, $g \leq o$ and $P = \infty$. Assume G has sink x and sources $y_1, \ldots, y_n$, such that $\mu(y_1) \leq \ldots \leq \mu(y_n)$. The following lemmas prove some properties of optimal schedules for (G, μ, c, L, o, g, P).

Lemma 4.1. *There is an optimal schedule (σ, π) for (G, μ, c, L, o, g, P), such that, for all sources y_{i_1} and y_{i_2}, if $\pi(y_{i_1}) = \pi(x)$ and $\pi(y_{i_2}) \neq \pi(x)$, then $\sigma(y_{i_1}) < \sigma(r(y_{i_2}, 1, j))$ for all $j \leq c(y_{i_2})$.*

Proof. Let (σ, π) be an optimal schedule for (G, μ, c, L, o, g, P). Consider two sources y_{i_1} and y_{i_2}, such that $\pi(y_{i_1}) = \pi(x)$ and $\pi(y_{i_2}) \neq \pi(x)$. Suppose $\sigma(y_{i_1}) > \sigma(r(y_{i_2}, 1, j))$ for some $j \leq c(y_{i_2})$. We may assume $\sigma(y_{i_1}) = \sigma(r(y_{i_2}, 1, j)) + o$. Then y_{i_1} can be scheduled at time $\sigma(r(y_{i_2}, 1, j))$, $r(y_{i_2}, 1, j)$ at time $\sigma(r(y_{i_2}, 1, j)) + \mu(y_{i_1})$ and $s(y_{i_2}, 1, j)$ at time $\sigma(r(y_{i_2}, 1, j)) + \mu(y_{i_1}) - o - L$ without violating the feasibility of (σ, π) or increasing its length. By repeating this step, an optimal schedule is constructed in which no source is scheduled after a receive task on the same processor. □

Lemma 4.2. *There is an optimal schedule (σ, π) for (G, μ, c, L, o, g, P), such that, for all processors $p \neq \pi(x)$, at most one source is executed on processor p.*

Proof. Let (σ, π) be an optimal schedule for (G, μ, c, L, o, g, P). We may assume $\pi(x) = 1$. Suppose two sources y_{i_1} and y_{i_2} are scheduled on processor $p \neq 1$. Let processor p' be a processor on which no task is executed. Then y_{i_2} can be scheduled on processor p' at time $\sigma(y_{i_2})$ and the send tasks $s(y_{i_2}, 1, j)$ on the same processor at times $\sigma(s(y_{i_2}, 1, j))$. This does not violate the feasibility of (σ, π) or increase the schedule length. Hence there is an optimal schedule for (G, μ, c, L, o, g, P), such that at most one source is executed on processor p for all $p \neq 1$. □

Lemma 4.3. *There is an optimal schedule (σ, π) for (G, μ, c, L, o, g, P), such that, for all sources y_{i_1} and y_{i_2}, if $i_1 < i_2$ and $\pi(y_i), \pi(y_{i_2}) \neq \pi(x)$, then $\sigma(r(y_{i_1}, 1, j_1)) < \sigma(r(y_{i_2}, 1, j_2))$ for all $j_1 \leq c(y_{i_1})$ and $j_2 \leq c(y_{i_2})$.*

Proof. Let (σ, π) be an optimal schedule for (G, μ, c, L, o, g, P). From Lemma 4.2, we may assume that $\pi(x) = 1$ and all processors $p \neq 1$ execute at most one task. Let y_{i_1} and y_{i_2} be two sources that are not scheduled on processor 1. Assume $i_1 < i_2$ and $\sigma(y_{i_1}) = \sigma(y_{i_2}) = 0$. Receive tasks $r(y_{i_1}, 1, j)$ can start at time $t_1 = \mu(y_{i_1}) + o + L$, receive tasks $r(y_{i_2}, 1, j)$ at time $t_2 = \mu(y_{i_2}) + o + L$. Then $t_1 \leq t_2$. Suppose $\sigma(r(y_{i_2}, 1, j_2)) < \sigma(r(y_{i_1}, 1, j_1))$ for some $j_1 \leq c(y_{i_1})$ and $j_2 \leq c(y_{i_2})$. Then $r(y_{i_2}, 1, j_2)$ can be scheduled at time $\sigma(r(y_{i_1}, 1, j_1))$ and $r(y_{i_1}, 1, j_1)$ at time $\sigma(r(y_{i_2}, 1, j_2))$. In addition, send task $s(y_{i_1}, 1, j_1)$ and $s(y_{i_2}, 1, j_2)$ can be scheduled $o + L$ time units before $r(y_{i_1}, 1, j_1)$ and $r(y_{i_2}, 1, j_2)$, respectively. This does not violate the feasibility of (σ, π) or increase its length, because all receive tasks have length o. So there is an optimal schedule in which, for all $i_1 < i_2$, receive tasks $r(y_{i_1}, 1, j_1)$ are scheduled before receive tasks $r(y_{i_2}, 1, j_2)$. □

Lemma 4.4. *There is an optimal schedule (σ, π) for (G, μ, c, L, o, g, P), such that for all sources y_i, if $\mu(y_i) \leq c(y_i)o$, then $\pi(y_i) = \pi(x)$.*

Proof. Let (σ, π) be an optimal schedule for (G, μ, c, L, o, g, P), such that $\pi(x) = 1$. From Lemmas 4.1 and 4.3, we may assume the sources on processor 1 are scheduled before the receive tasks of the sources scheduled on another processor and that the receive tasks of one source are scheduled on processor 1 without interruption. Suppose y_i is a source of G, such that $\mu(y_i) \leq c(y_i)o$ and $\pi(y_i) \neq 1$. Assume the first receive task $r(y_i, 1, j)$ is scheduled at time t. Then the last is completed at time $t + c(y_i)o \geq t + \mu(y_i)$. Then y_i can be rescheduled at time t on processor 1 without increasing the length of (σ, π) or violating its feasibility. Hence there is a schedule in which sources y_i with $\mu(y_i) \leq c(y_i)o$ are scheduled on the processor that executes x. □

Lemma 4.5. *A schedule for (G, μ, c, L, o, g, P) of length $\mu(x) + \sum_{i=1}^{n} \mu(y_i)$ is optimal if and only if, for all sources y_i, if $\mu(y_i) > c(y_i)o$, then $\sum_{j=1}^{n} \mu(y_j) \leq (c(y_i) + 1)o + L + \mu(y_i)$.*

Proof. ($\Rightarrow$) Suppose a schedule for (G, μ, c, L, o, g, P) of length $\mu(x) + \sum_{i=1}^{n} \mu(y_i)$ is optimal. Suppose there is a source y_i with $\mu(y_i) > c(y_i)o$ and $\sum_{j=1}^{n} \mu(y_j) > (c(y_i)+1)o+L+\mu(y_i)$. Then construct a schedule (σ, π) for (G, μ, c, L, o, g, P) as follows. Tasks $y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_n$ are scheduled on processor 1 from time 0 onward. y_i is scheduled on processor 2 at time 0. For $j \leq c(y_i)$, receive task $r(y_i, 1, j)$ is scheduled on processor 1 at time $\max\{\sum_{j \neq i} \mu(y_j), \mu(y_i) + jo + L\}$. For $j \leq c(y_i)$, send task $s(y_i, 1, j)$ is scheduled on processor 2 at time $\sigma(r(y_i, 1, j)) - L - o$. Then (σ, π) has length

$$\mu(x) + \max\{\mu(y_i) + (c(y_i) + 1)o + L, \sum_{j \neq i} \mu(y_j) + c(y_i)o\} \quad < \quad \mu(x) + \sum_{j=1}^{n} \mu(y_j).$$

Contradiction.

($\Leftarrow$) Suppose $\sum_{j=1}^{n} \mu(y_j) \leq (c(y_i) + 1)o + L + \mu(y_i)$ for all sources y_i with $\mu(y_i) > c(y_i)o$. Let (σ, π) be an optimal schedule for (G, μ, c, L, o, g, P). Suppose the length of (σ, π) is less than $\mu(x) + \sum_{i=1}^{n} \mu(y_i)$. From Lemma 4.4, we may assume that all tasks y_i with $\mu(y_i) \leq c(y_i)o$ are scheduled on processor $\pi(x)$. At least one source y_i is scheduled on another processor. Then $\mu(y_i) > c(y_i)o$. So (σ, π) has length at least

$$\mu(y_i) + (c(y_i) + 1)o + L + \mu(x) \quad \geq \quad \mu(x) + \sum_{i=1}^{n} \mu(y_i).$$

Contradiction.

□

These lemmas can be used to prove lower bounds on the length of an optimal schedule for (G, μ, c, L, o, g, P). Let ℓ^* be the length of an optimal schedule for (G, μ, c, L, o, g, P). Obviously, if $\ell^* < \mu(x) + \sum_{i=1}^{n} \mu(y_i)$, then $\ell^* \geq \mu(x) + 2o + L$. In addition, for each source y_i, either y_i itself or $c(y_i)$ receive tasks are scheduled on the same processor as x. Hence

$$\ell^* \quad \geq \quad \mu(x) + \sum_{i=1}^{n} \min\{\mu(y_i), c(y_i)o\}.$$

A schedule (σ, π) for (G, μ, c, L, o, g, P) will be constructed as follows. Sink x is scheduled on processor 1 and all sources y_i with $\mu(y_i) > c(y_i)o$ on a separate processor. The receive tasks are scheduled after the sources on processor 1, such that, if $i_1 < i_2$ and y_{i_1} and y_{i_2} are not scheduled on processor 1, then receive tasks $r(y_{i_1}, 1, j_{i_1})$ are executed before receive tasks $r(y_{i_2}, 1, j_{i_2})$. Assume sources $y_{i_1}, \ldots, y_{i_k}$ are the sources y_i with $\mu(y_i) > c(y_i)o$ and $i_1 < \ldots < i_k$ and let $y_{i_{k+1}}, \ldots, y_{i_n}$ be the remaining sources. If all tasks start as early as possible, then (σ, π) has length

$$\ell \quad \leq \quad \mu(x) + \max\{\sum_{j=1}^{k} c(y_{i_j})o + \sum_{j=k+1}^{n} \mu(y_{i_j}), \max_{1 \leq j \leq k} \mu(y_{i_j}) + \sum_{t=j}^{k} c(y_{i_t})o + 2o + L\}.$$

Assume ℓ^* is less than $\mu(x) + \sum_{i=1}^{n} \mu(y_i)$. In that case,

$$\begin{aligned}\ell &\leq \mu(x) + \max\{\sum_{j=1}^{k} c(y_{i_j})o + \sum_{j=k+1}^{n} \mu(y_{i_j}), \max_{1\leq j\leq k} \mu(y_{i_j}) + \sum_{t=j}^{k} c(y_{i_t})o + 2o + L\} \\ &\leq \max\{\ell^*, \ell^* + \ell^* + \ell^*\} \\ &= 3\ell^*.\end{aligned}$$

If the length of an optimal schedule for (G, μ, c, L, o, g, P) equals $\mu(x) + \sum_{i=1}^{n} \mu(y_i)$, then this can be checked in polynomial time using Lemma 4.5. In that case, we can construct an optimal schedule by scheduling all tasks on the same processor. Hence we have proved the following theorem.

Theorem 4.6. *Let (G, μ, c, L, o, g, P) be a LogP instance, such that G is a receive graph, $g \leq o$ and $P = \infty$. Then, in polynomial time, a schedule for (G, μ, c, L, o, g, P) can be constructed whose length is at most three times the length of an optimal schedule for (G, μ, c, L, o, g, P).*

4.2 A $4 - \frac{2}{P}$-approximation algorithm for the restricted case

The second approximation algorithm considers the case that the number of processors does not exceed the number of sources. Consider an instance (G, μ, c, L, o, g, P), such that G is a receive graph with sink x and sources $y_1, \ldots, y_n$, $g \leq o$ and $P < n$.

Like for the unrestricted case, the sources y_i with $\mu(y_i) \leq c(y_i)o$ will be scheduled on the same processor as x. Let Y_2 be the set of these sources and Y_1 the set of the remaining sources. Moreover, let $Y = Y_1 \cup Y_2$.

The sources of Y_1 are scheduled on processors $1, \ldots, P$ using Graham's List scheduling algorithm [15]. First set $idle(p) = 0$ for all processors p. For all tasks y in Y_1, y is scheduled as early as possible on a processor that becomes idle as early as possible. If y is scheduled on processor p, then $idle(p)$ is increased by $\mu(y)$.

Determine a processor p for which $\sum_{y\in Y_1:\pi(y)=p} c(y)$ is maximum. We can rearrange the processor assignment for the tasks of Y_1, such that $p = 1$. In addition, we may assume that $idle(2) \leq \ldots \leq idle(P)$. Then $\sum_{y\in Y_1:\pi(y)=1} c(y) \geq \frac{1}{P} \sum_{y\in Y_1} c(y)$. Schedule the tasks of Y_2 on processor 1 from time $idle(1)$ onward. Then $idle(1)$ is increased by $\sum_{y\in Y_2} \mu(y_2)$.

The communication tasks starting with those of the sources scheduled on processor 2 are scheduled as early as possible. $k_p = \sum_{y\in Y_1:\pi(y)=p} c(y)$ messages have to be sent from processor p to processor 1. Then the receive tasks are scheduled on processor 1 from time $\max\{idle(1), idle(p)+o+L\}$ onward. The send tasks are scheduled on processor p from $\max\{idle(1), idle(p) + o + L\} - o - L$ onward. After that $idle(1)$ equals $\max\{idle(1), idle(p) + o + L\} + k_p o$.

Let (σ, π) be the resulting schedule. Assume the length of (σ, π) is ℓ. Let ℓ^* be the length of an optimal schedule for (G, μ, c, L, o, g, P). Then, similar to the unrestricted case,

$$\ell^* \geq \mu(x) + \sum_{i=1}^{n} \min\{\mu(y_i), c(y_i)o\} = \mu(x) + \sum_{y\in Y_2} \mu(y) + \sum_{y\in Y_1} c(y)o.$$

In addition, $\ell^* \geq \mu(x) + \frac{1}{P} \sum_{i=1}^{n} \mu(y_i)$. We will assume the schedule in which all tasks are scheduled on one processor is not optimal. In that case, $\ell^* \geq \mu(x) + L + 2o$.

Let y^* be a source of Y_1 with a maximum completion time. It is possible that every task in Y_2 is scheduled after y^*. Hence

$$\ell \leq \sigma(y^*) + \mu(y^*) + \sum_{y\in Y_2} \mu(y) + \sum_{y\in Y_1:\pi(y)\neq 1} c(y)o + o + L + \mu(x).$$

All processors are completely filled until time $\sigma(y^*)$, otherwise, y^* would have been scheduled at an earlier time. Hence

$$\sigma(y^*) \leq \frac{1}{P} \sum_{y\in Y_1\setminus\{y^*\}} \mu(y).$$

So y^* is completed at time

$$\begin{aligned}\sigma(y^*)+\mu(y^*) &\leq \frac{1}{P}\sum_{y\in Y_1\setminus\{y^*\}}\mu(y)+\mu(y^*)\\ &= \frac{1}{P}\sum_{y\in Y_1}\mu(y)+(1-\frac{1}{P})\mu(y^*).\end{aligned}$$

Therefore x is completed at time

$$\begin{aligned}\ell &\leq \sigma(y^*)+\mu(y^*)+\sum_{y\in Y_2}\mu(y)+\sum_{y\in Y_1:\pi(y)\neq 1}c(y)o+o+L+\mu(x)\\ &\leq \frac{1}{P}\sum_{y\in Y_1}\mu(y)+(1-\frac{1}{P})\mu(y^*)+\frac{1}{P}\sum_{y\in Y_2}\mu(y)+\\ &\qquad (1-\frac{1}{P})\sum_{y\in Y_2}\mu(y)+(1-\frac{1}{P})\sum_{y\in Y_1}c(y)o+o+L+\mu(x)\\ &\leq \ell^*+(1-\frac{1}{P})\ell^*+(1-\frac{1}{P})\ell^*+\ell^*\\ &= (4-\frac{2}{P})\ell^*.\end{aligned}$$

If the schedule in which all tasks are executed on one processor is optimal, then its length is at most ℓ. If (σ,π) is longer than $\mu(x)+\sum_{i=1}^{n}\mu(y_i)$, then replace (σ,π) by the schedule in which all tasks are executed by the same processor. Then this schedule is at most $4-\frac{2}{P}$ times as long as an optimal schedule. Hence we have proved the following theorem.

Theorem 4.7. *Let (G,μ,c,L,o,g,P) be a LogP instance, such that G is a receive graph and $g\leq o$. Then, in polynomial time, a schedule for (G,μ,c,L,o,g,P) can be constructed whose length is at most $4-\frac{2}{P}$ times the length of an optimal schedule for (G,μ,c,L,o,g,P).*

5 d-ary intrees

In this section, we consider the problem of scheduling d-ary intrees in the LogP model. An approximation algorithm is presented that schedules d-ary intrees. The basis of the algorithm is an algorithm that constructs decompositions of d-ary intrees into subforests whose sizes do not differ much. Such a decomposition is used to construct a schedule in which the communication requirements are ignored. The necessary communication tasks are introduced in this communication-free schedule.

This algorithm is presented with three types of decompositions; one for full d-ary intrees with unit-length tasks, and two for arbitrary d-ary intrees with arbitrary task lengths.

Consider a LogP instance (T,μ,c,L,o,g,P), such that T is an intree. A *decomposition* of (T,μ,c,L,o,g,P) is a collection of subforests $T_1,\ldots,T_k$ of T, such that the roots of T_i all have the same parent, a task in T_i has no predecessors in $T_{i+1},\ldots,T_k$ and every task of T is an element of exactly one subforest T_i.

Using a decomposition $T_1,\ldots,T_k$ of (T,μ,c,L,o,g,P), we can construct a schedule. This is done in two steps. First, a *communication-free schedule* is constructed. A communication-free schedule is a schedule in which no precedence constraint is violated and no processor executes two tasks at the same time. No send and receive tasks are scheduled in such a schedule. Second, the communication operations are introduced in the communication-free schedule for (T,μ,c,L,o,g,P).

Consider a communication-free schedule (σ,π) for (T,μ,c,L,o,g,P). The *communication requirement* of (σ,π) equals the number of pairs of tasks (v_1,v_2), such that $v_1\prec_0 v_2$ and $\pi(v_1)\neq\pi(v_2)$. Such a pair of tasks will be called a *communicating pair*. It is not difficult to transform a

communication-free schedule (σ, π) into a feasible schedule for (T, μ, c, L, o, g, P): this is accomplished by adding communication operations between every communicating pair (v_1, v_2). Then the length of the resulting schedule exceeds that of the communication-free schedule by the total duration of the communication operations.

Constructing communication-free schedules corresponds to scheduling without communication delays and overheads. Kunde [19] proved that, for inforests, critical path scheduling constructs (communication-free) schedules of length at most $2 - \frac{2}{P+1}$ times the length of an optimal schedule. Unfortunately, the communication requirement of such a schedule may be as high as $(1 - \frac{1}{d})n + \frac{1}{d}$ for d-ary intrees with n tasks. As a result, introducing communication operations in such a schedule may greatly increase the length of the schedule. The decomposition algorithms that will be presented in this section allow us to construct communication-free schedules that are longer than those constructed by critical path scheduling, but their communication requirement depends only on the maximum indegree and the number of processors.

The decompositions are used by the approximation algorithm to construct a schedule. First some notations are introduced. Let (T, μ, c, L, o, g, P) be a LogP instance, such that T is a d-ary intree. Let $T_1, \ldots, T_k$ be a decomposition of (T, μ, c, L, o, g, P). Since all roots of a subforest T_i have the same parent and no task in $T_1, \ldots, T_{k-1}$ has a predecessor in T_k, T_k is a subtree of T that contains the root r of T. Assume $T_i = T_{i,1} \cup \ldots \cup T_{i,n_i}$, such that $T_{i,j}$ is one of the trees of T_i. Let $r_{i,j}$ be the root of $T_{i,j}$. Note that $T_k = T_{k,1}$ and $r_{k,1}$ is the root of T. Let $n = \sum_{i=1}^{k} n_i$.

Define $PT = \bigcup_{i=1}^{k} p(r_{i,1}, r_{k,1})$ as the intree containing the tasks of the paths from the parents of the roots of $T_{i,j}$ to the root of T. Let u be a task of T_{i_1} with a predecessor in T_{i_2} with $i_2 \neq i_1$. Then $u \in p(r_{i_2,1}, r_{k,1})$. So PT contains the tasks in the trees $T_{i,j}$ with predecessors in $T \setminus T_{i,j}$. PT is a subtree of T and the root of PT is the root of T.

Let u be a task of PT that has children outside PT. Then at least one child of u is the root of a decomposition tree $T_{i,j}$. The root of T is one of these roots, so the number of tasks in PT with predecessors outside PT is at most $n - 1$. As a result, the total number of tasks outside PT that are children of tasks of PT is at most $d(n-1)$.

Consider the tree PT^* consisting of the tasks of PT and their children. Let u be a task of PT^*. Define $d^*(u)$ as the number of children of u in PT^*. Note that if $d^*(u) \geq 1$, then u is a task of PT. Assume PT^* contains m tasks, m_1 of which are elements of PT and $m_0 = m - m_1$ are sources of PT^*. Then

$$\begin{aligned} \sum_{u \in PT} (indegree(u) - 1) &= -m_1 + \sum_{u \in PT^*} d^*(u) \\ &= -m_1 + m - 1 \\ &= m_0 - 1 \\ &\leq d(n-1) - 1. \end{aligned}$$

Note that if the only tasks outside PT that are children of tasks are roots of decomposition trees, then $\sum_{u \in PT}(indegree(u) - 1) \leq n - 1$.

From a decomposition $T_1, \ldots, T_k$, we can construct a decomposition tree DT. DT contains n tasks: tasks $r_{i,j}$, such that $1 \leq i \leq k$ and $1 \leq j \leq n_i$. There is an arc from r_{i_1,j_2} to r_{i_2,j_2} if, in T, the parent of r_{i_1,j_1} is a task of T_{i_2,j_2}. If r_{i_1,j_1} is a predecessor of r_{i_2,j_2} in DT, then this is denoted by $r_{i_1,j_1} \prec_D r_{i_2,j_2}$. Similarly, if r_{i_1,j_1} is a child of r_{i_2,j_2} in DT, then we denote this by $r_{i_1,j_1} \prec_{D,0} r_{i_2,j_2}$. To make a difference between precedence constraints in T and DT, the terms *DT-child*, *DT-parent*, *DT-successor* and *DT-predecessor* will be used.

Using a decomposition $T_1, \ldots, T_k$ of an instance (T, μ, c, L, o, g, P), such that T is an intree and $k \leq P$, Algorithm DECOMPOSITION TREE SCHEDULING constructs a communication-free schedule for (T, μ, c, L, o, g, P). $idle(i)$ denotes the earliest time after which processor i remains empty.

Let (σ, π) be the communication-free schedule for (T, μ, c, L, o, g, P) constructed by Algorithm DECOMPOSITION TREE SCHEDULING using decomposition $T_1, \ldots, T_k$. Let $C(r_{i,j}) = \sigma(r_{i,j}) +$

Algorithm DECOMPOSITION TREE SCHEDULING
Input: A LogP instance (T, μ, c, L, o, g, P), such that T is an intree and a decomposition of (T, μ, c, L, o, g, P) consisting of $k \leq P$ subforests $T_1, \ldots, T_k$.
Output: A communication-free schedule (σ, π) for (T, μ, c, L, o, g, P).

```
1.  for i := 1 to k
2.      do let v_1, ..., v_r be a topological ordering of T_i \ PT
3.         idle(i) := 0
4.         for j := 1 to r
5.             do σ(v_j) := idle(i)
6.                π(v_j) := i
7.                idle(i) := idle(i) + μ(v_j)
8.         let w_1, ..., w_s be a topological order of T_i ∩ PT
9.         for j := 1 to s
10.            do assume the last child of w_j outside T_i \ PT is scheduled on processor p
11.               σ(w_j) := max{idle(p), idle(i)}
12.               π(w_j) := p
13.               idle(p) := σ(w_j) + μ(w_j)
```

Figure 6: The algorithm constructing communication-free schedules

$\mu(r_{i,j})$ be the completion time of $r_{i,j}$ and C_i the maximum completion time of a task of T_i. If a task of $T_{i,j} \cap PT$ has a child outside of $T_{i,j}$, then this child is a root of a decomposition tree $T_{i',j'}$ with $i' < i$.

Assume r_{i_1,j_1} and r_{i_2,j_2} are DT-children of $r_{i,j}$ and $i_2 > i_1$. If r_{i_2,j_2} is scheduled on the same processor as r_{i_1,j_1}, then r_{i_2,j_2} is a successor of r_{i_1,j_1}. In that case, r_{i_2,j_2} is not a DT-child of $r_{i,j}$. So all DT-children r_{i_1,j_1} and r_{i_2,j_2} with $i_1 \neq i_2$ of $r_{i,j}$ are scheduled on different processors. Hence, for all DT-children $r_{i',j'}$ of $r_{i,j}$, $idle(\pi(r_{i',j'}))$ equals C_j before the tasks of $T_i \cap PT$ are scheduled.

Let v be a task of $T_{i,j} \cap PT$ that has a predecessor that is not an element of $T_{i,j} \setminus PT$. Then v is scheduled on the first idle time slot (not before time $idle(i)$) on the processor that executes the last predecessor of v that is not an element of $T_{i,j} \setminus PT$. Hence, for all $1 \leq i \leq k$ and $1 \leq j \leq n_i$,

$$\begin{aligned} C(r_{i,j}) &\leq \max\{\mu(T_i \setminus PT) + \max_{r_{i',j'} \prec_{D,0} r_{i,j}} \mu(p(r_{i',j'}, r_{i,j})), \max_{r_{i',j'} \prec_{D,0} r_{i,j}} C_{i'} + \mu(p(r_{i',j'}, r_{i,j}))\} \\ &\leq \max\{\mu(T_i), \max_{r_{i',j'} \prec_{D,0} r_{i,j}} C_{i'} + \mu(p(r_{i',j'}, r_{i,j}))\}. \end{aligned}$$

Consequently,

$$C_i \leq \max\{\mu(T_i), \max_{r_{i',j'} \prec_{D,0} r_{i,j}} C_{i'} + \mu(p(r_{i',j'}, r_{i,j}))\}.$$

Using induction, we can prove that, for all $i \leq k$,

$$C_i \leq \max\{\mu(T_i), \max_{r_{i',j'} \prec_{D} r_{i,j}} \mu(T_{i'}) + \mu(p(r_{i',j'}, r_{i,j}))\}.$$

Since $r_{k,1}$ is the root of T and $n_k = 1$ and all roots of a decomposition forest T_j have the same parent, the length of (σ, π) is at most

$$\max\{\mu(T_k), \max_{i<k} \mu(T_i) + \mu(p(r_{i,1}, r_{k,1}))\}.$$

So we have proved the following lemma.

Lemma 5.1. *Let (T, μ, c, L, o, g, P) be a LogP instance, such that T is an intree. Let $T_1, \ldots, T_k$ be a decomposition of (T, μ, c, L, o, g, P), such that $k \leq P$. Then, using $T_1, \ldots, T_k$, Algorithm* DECOMPOSITION TREE SCHEDULING *constructs a communication-free schedule for (T, μ, c, L, o, g, P) of length at most*

$$\max\{\mu(T_k), \max_{i<k} \mu(T_i) + \mu(p(r_{i,1}, r_{k,1}))\}.$$

Consider the communication-free schedule (σ, π) for (T, μ, c, L, o, g, P) constructed by Algorithm DECOMPOSITION TREE SCHEDULING. Let v be a task of T. Let $d(v)$ be the indegree of v. If a predecessor of v is scheduled on a different processor, then v is an element of PT and at most $d(v) - 1$ children of v are executed on different processors. Hence there are at most $d(n-1) - 1$ communication operations in the feasible schedule (σ_c, π_c) constructed from (σ, π) by introducing the communication tasks. Let $c_{\max} = \max_u c(u)$. Then the length of (σ_c, π_c) is at most the length of (σ, π) plus $(d(n-1)-1)(L+o+c_{\max}\max\{o, g\})$. Hence we have proved the following theorem.

Theorem 5.2. *Let (T, μ, c, L, o, g, P) be a LogP instance, such that T is a d-ary intree. Let $T_1, \ldots, T_k$ be a decomposition of (T, μ, c, L, o, g, P), such that $k \leq P$. Then, in polynomial time, a schedule for (T, μ, c, L, o, g, P) of length at most*

$$\max\{\mu(T_k), \max_{i<k} \mu(T_i) + \mu(p(r_{i,1}, r_{k,1}))\} + (d(n-1)-1)(L+o+c_{\max}\max\{o, g\})$$

can be constructed, where n is the number of roots of $T_1, \ldots, T_k$.

Note that if the roots of the decomposition trees are the only tasks outside PT that are children of tasks of PT, then at most $n-1$ communication operations are necessary.

5.1 An approximation algorithm for full d-ary intrees

In this section, we will consider full d-ary intrees with unit-length tasks. We will construct decompositions of such intrees. The decompositions are used by Algorithm DECOMPOSITION TREE SCHEDULING to construct a schedule.

An instance of a corresponding scheduling problem with unit-length tasks is denoted by $(T, \mathbf{1}_T, c, L, o, g, P)$. Assume T is a full d-ary intree of height h. Then the longest path in T contains h tasks. Let r be the root of T. A task u of T is said to be of *level* k if the path from u to r contains k tasks. Note that there are d^k tasks of level k in T. Hence the total number of tasks in T equals $\sum_{i=0}^{h-1} d^i = \frac{d^h-1}{d-1}$.

Determine k, such that $d^k \leq P < d^{k+1}$. Let $m = \lfloor \frac{P}{d^k} \rfloor d^k$. Then m is the smallest multiple of d^k that does not exceed P. Since $m \geq d^k$, $P < 2m$. We will construct a decomposition of $(T, \mathbf{1}_T, c, L, o, g, P)$ consisting of m subtrees of T.

There are d^k tasks of level k. Assume $u_1, \ldots, u_{d^k}$ are the tasks of level k. Let $s = \frac{m}{d^k}$. Consider a task u_i of level k. Subtree $T(u_i)$ of T will be decomposed into s subforests $T_{i,1}, \ldots, T_{i,s}$ of $T(u_i)$. Let $v_1, \ldots, v_d$ be the children of u_i. Let $p = \lfloor \frac{d}{s} \rfloor$ and $q = d - \lfloor \frac{d}{s} \rfloor s$. Then $q(p+1) + (s-q)p$ equals d. We will construct q subforests consisting of $p+1$ trees $T(v_j)$ and $s-q$ that consist of p subtrees $T(v_j)$. For $1 \leq j \leq q$, let

$$T_{i,j} \quad = \quad \bigcup_{t=(j-1)(p+1)+1}^{j(p+1)} T(v_t)$$

and, for $q+1 \leq j \leq s-1$, let

$$T_{i,j} \quad = \quad \bigcup_{t=(j-1)p+q+1}^{jp+q} T(v_t) \quad \text{and} \quad T_{i,s} \quad = \quad \bigcup_{t=(s-1)p+q+1}^{d} T(v_t) \cup \{u_i\}.$$

Now all tasks of level at least k are an element of some tree $T_{i,j}$. The remaining tasks are added to the trees as follows. For $1 \leq j \leq d^k$, add the tasks of $p(u_j, r) \setminus \bigcup_{t=j+1}^{d^k} p(u_t, r)$ to subforest $T_{j,s}$. Then no task of $T_{i_2,s}$ is a predecessor of a task of $T_{i_1,s}$ if $i_1 < i_2$. Moreover, a task of T_{i,j_2} is not a predecessor of a task in T_{i,j_1} if $j_1 < j_2$. In addition, the roots of a forest $T_{i,j}$ have the same parent.

Consider the resulting decomposition $T_{1,1}, \ldots, T_{1,s}, \ldots, T_{d^k,1}, \ldots, T_{d^k,s}$. If d is divisible by s, then q equals 0, $p = \lceil \frac{d}{s} \rceil$ and each forest has at most $k+1+p\sum_{i=k+1}^{h-1} d^{i-k-1}$ elements. Otherwise,

$\lceil \frac{d}{s} \rceil = p+1$ and each decomposition forest consists of at most $k+1+(p+1)\sum_{i=k+1}^{h-1} d^{i-k-1}$ tasks. In either case, a decomposition forest contains at most $k+1+\lceil \frac{d}{s} \rceil \sum_{i=k+1}^{h-1} d^{i-k-1}$ tasks. From Lemma 5.1, Algorithm DECOMPOSITION TREE SCHEDULING constructs a communication-free schedule (σ, π) for $(T, \mathbf{1}_T, c, L, o, g, P)$ of length at most

$$k+1+\left\lceil \frac{d}{s} \right\rceil \sum_{i=k+1}^{h-1} d^{i-k-1}.$$

Let ℓ be the length of (σ, π).

Let ℓ^* be the length of an optimal schedule for $(T, \mathbf{1}_T, c, L, o, g, P)$. For each $k^* \leq h-1$, there are d^{k^*} tasks of level k^*. Let u be a task of level k^*. Then $T(u)$ contains $\sum_{i=k^*}^{h-1} d^{i-k^*}$ tasks. The path from u to the root of T has length k^*. As a result, for all $k^* \leq h-1$,

$$\ell^* \quad \geq \quad k^* + \frac{d^{k^*}}{P} \sum_{i=k^*}^{h-1} d^{i-k^*}.$$

As a result,

$$\begin{aligned}
\ell &\leq k+1+\left\lceil \frac{d}{s} \right\rceil \sum_{i=k+1}^{h-1} d^{i-k-1} \\
&= k+1+\left\lceil \frac{d}{\frac{m}{d^k}} \right\rceil \sum_{i=k+1}^{h-1} d^{i-k-1} \\
&= k+1+\left\lceil \frac{d^{k+1}}{m} \right\rceil \sum_{i=k+1}^{h-1} d^{i-k-1} \\
&\leq k+1+\frac{P}{m}\frac{d^{k+1}}{P} \sum_{i=k+1}^{h-1} d^{i-k-1} + \sum_{i=k+1}^{h-1} d^{i-k-1} \\
&\leq \frac{P}{m}(k+1+\frac{d^{k+1}}{P} \sum_{i=k+1}^{h-1} d^{i-k-1}) + \frac{P}{d^{k+1}}\frac{d^{k+1}}{P} \sum_{i=k+1}^{h-1} d^{i-k-1} \\
&\leq \frac{P}{m}\ell^* + \frac{P}{d^{k+1}}\ell^* \\
&= (\frac{P}{m} + \frac{P}{d^{k+1}})\ell^*.
\end{aligned}$$

Let $x = \lfloor \frac{P}{d^k} \rfloor$. Then $m = d^k x$, $P < d^k(x+1)$ and $1 \leq x \leq d-1$. Therefore,

$$\begin{aligned}
\ell &\leq (\frac{P}{m} + \frac{P}{d^{k+1}})\ell^* \\
&\leq (\frac{d^k(x+1)}{d^k x} + \frac{d^k(x+1)}{d^{k+1}})\ell^* \\
&= (\frac{x+1}{x} + \frac{x+1}{d})\ell^*.
\end{aligned}$$

Now consider the function

$$f_d : x \to \frac{x+1}{x} + \frac{x+1}{d}$$

for $1 \leq x \leq d-1$. Let f'_d be the derivative of f_d. Then

$$f'_d(x) = \frac{1}{x^2} - \frac{1}{d}.$$

$f'_d(x)$ is negative for $1 \leq x < \sqrt{d}$, zero for $x = \sqrt{d}$ and positive for $\sqrt{d} < x \leq d-1$. Hence $f_d(x) \leq \max\{f_d(1), f_d(d-1)\}$ for all x. Hence

$$\ell \quad \leq \quad \max\{2+\frac{2}{d}, 2+\frac{1}{d-1}\}\ell^* \quad = \quad (2+\frac{2}{d})\ell^*.$$

Note that if d equals two, then $2^k \leq P < 2^{k+1}$. Hence $m = \lfloor \frac{P}{2^k} \rfloor 2^k = 2^k$. In that case, 2^{k+1} is divisible by m and hence $\ell \leq \frac{P}{m}\ell^* < 2\ell^*$.

The resulting communication-free schedule can be transformed into a feasible schedule for $(T, \mathbf{1}_T, c, L, o, g, P)$ by introducing the communication operations. The number of communication operations equals the total number of roots of the decomposition trees $T_{i,j}$, because PT consists of all tasks of level at most $k-1$. Consider the forests $T_{i,1}, \ldots, T_{i,s}$. Each of these forests consists of at most $\lceil \frac{d}{s} \rceil = \lceil \frac{d^{k+1}}{m} \rceil$ trees. Hence the total number of roots is at most

$$\begin{aligned} d^k s \left\lceil \frac{d^{k+1}}{m} \right\rceil &\leq d^k \frac{m}{d^k} \frac{d^{k+1}}{m} + d^k \frac{m}{d^k} \\ &= d^{k+1} + m \\ &\leq (d+1)P. \end{aligned}$$

Note that if d equals two, then the number of roots is at most $2P$. So we have proved the following theorem.

Theorem 5.3. *Let $(T, \mathbf{1}_T, c, L, o, g, P)$ be a LogP instance, such that T is a full d-ary intree. Let ℓ^* be the length of an optimal schedule for $(T, \mathbf{1}_T, c, L, o, g, P)$. In polynomial time, a schedule for $(T, \mathbf{1}_T, c, L, o, g, P)$ of length at most*

$$(2 + \frac{2}{d})\ell^* + ((d+1)P - 1)(L + o + c_{\max} \max\{o, g\})$$

can be constructed. If T is a full binary intree, then a schedule of length at most

$$2\ell^* + (2P - 1)(L + o + c_{\max} \max\{o, g\})$$

can be constructed in polynomial time.

5.2 An approximation algorithm for arbitrary d-ary intrees

In this section, we will construct decompositions for instances (T, μ, c, L, o, g, P), such that T is an arbitrary d-ary intree. These will be used by Algorithm DECOMPOSITION TREE SCHEDULING to construct a schedule for (T, μ, c, L, o, g, P). Consider an instance (T, μ, c, L, o, g, P), where T is a d-ary intree. Let N be the sum of the task lengths of the tasks of T. Let $1 \leq \beta \leq N$. (T, μ, c, L, o, g, P) is called *β-restricted* if every task of T has length at most β and every task with at least two children has unit length.

If (T, μ, c, L, o, g, P) is not β-restricted, then a β-restricted instance $(T_\beta, \mu_\beta, c_\beta, L, o, g, P)$ can be constructed as follows. Let u be a task of T. Assume $\mu(u) = k_1\beta + k_2 + 1$, such that $0 \leq k_2 \leq \beta - 1$. Then u is replaced by a chain of $k_1 + 2$ tasks $u_0 \prec u_1 \prec \ldots \prec u_k \prec u_{k+1}$, such that $\mu_\beta(u_0) = 1$, $\mu_\beta(u_1) = \ldots = \mu_\beta(u_{k_1}) = \beta$ and $\mu_\beta(u_{k_1+1}) = k_2$. In addition, $c_\beta(u_{k_1+1}) = c(u)$ and $c_\beta(u_0) = \ldots = c_\beta(u_{k_1}) = 0$. Note that if $k_2 = 0$, then u_{k_1+1} does not exist. In that case, $c_\beta(u_{k_1}) = c(u)$. Then $(T_\beta, \mu_\beta, c_\beta, L, o, g, P)$ is a β-restricted instance. The number of tasks of T_β is at most $\frac{N}{\beta} + 2$ times the number of tasks of T.

Let (T, μ, c, L, o, g, P) be a β-restricted instance, such that T is a d-ary intree. The next lemma allows the decomposition of (T, μ, c, L, o, g, P), such that the total task length of these intrees does not differ much. By $\mu(T)$, we denote the sum of the task length of the task of T. The following lemma is similar to a lemma of Kosaraju [18] that considers the number of leafs of binary trees.

Lemma 5.4. *Let (T, μ, c, L, o, g, P) be a β-restricted LogP instance, such that T is a d-ary intree and $\mu(T) \geq \beta$. Then T contains a task u, such that $\beta \leq \mu(T(u)) \leq d(\beta - 1) + 1$.*

Proof. This is obvious if T contains only one task. Suppose the lemma holds for all d-ary intrees with at most $n-1$ tasks and $n \geq 2$. Let (T, μ, c, L, o, g, P) be a β-restricted instance, such that T is a d-ary intree with n tasks and $\beta \leq \mu(T)$. Assume u is the root of T.

Case 1. u has indegree one.
Let v be the child of u. If $\mu(T(v)) \leq \beta - 1$, then $\beta \leq \mu(T(u)) \leq \mu(u) + \mu(T(v)) \leq 2\beta - 1 \leq d(\beta - 1) + 1$. Otherwise, with induction, $T(v)$ contains a task w, such that $\beta \leq \mu(T(w)) \leq d(\beta - 1) + 1$.

Case 2. u has indegree at least two.
Then u is a task of length one. If $\mu(T(v)) \leq \beta - 1$ for all children v of u, then $\beta \leq \mu(T(u)) \leq d(\beta - 1) + 1$. Otherwise, u has a child v, such that $\mu(T(v)) \geq \beta$. With induction, $T(v)$ contains a task w, such that $\beta \leq \mu(T(w)) \leq d(\beta - 1) + 1$.

□

Using this result, Algorithm d-ARY TREE DECOMPOSITION constructs decompositions of β-restricted instances (T, μ, c, L, o, g, P), such that T is a d-ary tree in at most k trees by repeatedly removing a task and its predecessors.

Algorithm d-ARY TREE DECOMPOSITION
Input: A β-restricted instance (T, μ, c, L, o, g, P), such that T is a d-ary intree with n nodes and an integer k with $1 \leq k \leq n$.
Output: A decomposition $T_1, \ldots, T_{k'}$ of (T, μ, c, L, o, g, P), such that $k' \leq k$ and, for all $i \leq \min\{k - 1, k'\}$, $\beta \leq \mu(T_i) \leq d(\beta - 1) + 1$.

```
1.  i := 1
2.  while μ(T) > d(β − 1) + 1 and i < k
3.      do let u_i be a task of T, such that β ≤ μ(T(u_i)) ≤ d(β − 1) + 1
4.          T_i := T(u_i)
5.          T := T \ T_i
6.          i := i + 1
7.  T_i := T
```

Figure 7: The decomposition algorithm for arbitrary d-ary intrees

The decompositions constructed by Algorithm d-ARY TREE DECOMPOSITION are used by the approximation algorithm. Note that every forest T_i of a decomposition constructed by Algorithm d-ARY TREE DECOMPOSITION is a tree. Hence the number of roots of such a decomposition equals the number of forests. Assume r_i is the root of T_i.

First we will prove some properties of decompositions of β-restricted instances. Consider a β-restricted instance (T, μ, c, L, o, g, P) constructed from original instance $(T^*, \mu^*, c^*, L, o, g, P)$. Let $T_1, \ldots, T_k$ be a decomposition of (T, μ, c, L, o, g, P) constructed by Algorithm d-ARY TREE DECOMPOSITION.

Lemma 5.5. *Let u be a task of T^* that is replaced by $u_1, \ldots, u_r$ in T. If $u_1, \ldots, u_r \in T_i$ and $u_j \in T_i \cap PT$, then $u_1, \ldots, u_r \in T_i \cap PT$.*

Proof. Suppose $u_1, \ldots, u_r \in T_i$ and $u_j \in T_i \cap PT$. u_j has a predecessor outside T_i. Since $u_{j+1}, \ldots, u_r$ are successors of u_j, they have a predecessor outside T_i as well. Let v be a predecessor of u_j outside T_i. Then $u_1, \ldots, u_{j-1}$ are tasks on the path from v to u_j, because $u_2, \ldots, u_j$ all have indegree one. So all tasks $u_1, \ldots, u_r$ are a successor of v. Hence these are all elements of $T_i \cap PT$. □

Lemma 5.6. *Let u be a task of T^* that is replaced by $u_1, \ldots, u_r$ in T. If $u_{j+1} \in T_i$ and $u_j \notin T_i$, then u_j is the root of a decomposition tree.*

Proof. Suppose $u_{j+1} \in T_i$ and $u_j \notin T_i$. Then u_{j+1} is a task of T_i with a child outside T_i. The children outside T_i of such a task are roots of decomposition trees. So u_j is the root of a decomposition tree. □

Assume (T, μ, c, L, o, g, P) is constructed by replacing tasks of length at least two from the instance $(T^*, \mu^*, c^*, L, o, g, P)$. A topological order $v_1, \ldots, v_k$ of a subtree T' of T is called T^*-*consistent* if, for all tasks u of T^* and all subtasks u_i of u, if u_i and u_{i+1} are elements of T', then $u_i = v_j$ and $u_{i+1} = v_{j+1}$ for some j. Clearly, for every subtree T' of T, there is a T^*-consistent topological order. We will assume Algorithm DECOMPOSITION TREE SCHEDULING uses such topological orders.

Let (σ, π) be the communication-free schedule for β-restricted instance (T, μ, c, L, o, g, P) constructed by Algorithm DECOMPOSITION TREE SCHEDULING using decomposition $T_1, \ldots, T_k$. From Lemma 5.1, the length of (σ, π) equals $\max\{\mu(T_k), \max_{j<k} \mu(T_j) + \mu(p(r_j, r_k))\}$.

(σ, π) need not be a communication-free schedule for (T, μ, c, L, o, g, P). It is, however, possible to built a communication-free schedule for (T, μ, c, L, o, g, P) with the same length as (σ, π). The following lemmas are used to prove this.

Lemma 5.7. *Let u be a task of T^* that is replaced by $u_1, \ldots, u_r$ in T. Then $\pi(u_1) = \ldots = \pi(u_r)$.*

Proof. Suppose $\pi(u_1) = \ldots = \pi(u_{j-1})$. Consider u_j. If u_j is an element of $T_i \setminus PT$ for some i, then u_{j-1} is an element of $T_i \setminus PT$ as well. In that case, u_{j-1} and u_j are scheduled on the same processor. If u_j is an element of $T_i \cap PT$ for some i, then u_{j-1} is the last predecessor of u_j. So u_j is scheduled on the same processor as u_{j-1}. □

Lemma 5.8. *Let u be a task of T^* that is replaced by $u_1, \ldots, u_r$ in T. Then processor $\pi(u_1)$ does not execute any tasks between u_j and u_{j+1}.*

Proof. Consider two tasks u_j and u_{j+1}, such that $\sigma(u_{j+1}) > \sigma(u_j) + \mu(u_j)$. Since Algorithm DECOMPOSITION TREE SCHEDULING uses T^*-consistent topological orders, u_j is an element of $T_i \cap PT$ for some i. All tasks scheduled on processor i after u_{j-1} are successors of u_{j-1}. Since u_j is the only immediate successor of u_{j-1}, no task is scheduled on processor $\pi(u_j)$ between u_{j-1} and u_j. □

Using Lemmas 5.7 and 5.8, it is easy to transform a communication-free schedule (σ, π) for (T, μ, c, L, o, g, P) into a communication-free schedule for $(T^*, \mu^*, c^*, L, o, g, P)$. Consider a task u of T^* that is replaced by $u_1, \ldots, u_r$. Assume $\sigma(u_{j+1}) = \sigma(u_j) + \mu(u_j)$ for all $j \geq i$. Then set $\sigma(u_{i-1}) = \sigma(u_i) - \mu(u_{i-1})$. Then there is no idle time between u_{i-1} and u_i. By repeating this, we get a schedule (σ^*, π^*) that coincides with a schedule (σ^*, π^*) for $(T^*, \mu^*, c^*, L, o, g, P)$: task u is scheduled on processor $\pi^*(u_1)$ at time $\sigma^*(u_1)$. The length of (σ^*, π^*) equals that of (σ, π). Hence we can construct a communication-free schedule for $(T^*, \mu^*, c^*, L, o, g, P)$ of length at most $\max\{\mu(T_k), \max_{j<k} \mu(T_j) + \mu(p(r_j, r_k))\}$. Moreover, since no task is moved to another processor, the communication requirement of the schedule for (T, μ, c, L, o, g, P) equals that of the schedule for (T, μ, c, L, o, g, P).

Consider the decomposition $T_1, \ldots, T_k$ of β-restricted instance (T, μ, c, L, o, g, P) constructed by Algorithm d-ARY TREE DECOMPOSITION using parameter P. Then $\beta \leq \mu(T_i) \leq d(\beta - 1) + 1$ for all $i \leq k - 1$. If $k \leq P - 1$, then $\beta \leq \mu(T_k) \leq d(\beta - 1) + 1$. Otherwise,

$$N - (P-1)(d(\beta-1)+1) \quad \leq \quad \mu(T_k) \quad \leq \quad N - (P-1)\beta.$$

Let (σ, π) be the communication-free schedule for (T, μ, c, L, o, g, P) constructed by Algorithm DECOMPOSITION TREE SCHEDULING using this decomposition. Let ℓ be the length of (σ, π) and ℓ^* the length of a minimum-length (communication-free) schedule for (T, μ, c, L, o, g, P). Then, for all $i \leq k$,

$$\ell^* \quad \geq \quad \frac{1}{P}\mu(T_i) + \mu(p(r_i, r_k)) \quad \text{and} \quad \ell^* \quad \geq \quad \frac{N}{P}.$$

We know that $\ell = C(r_k) \leq \max\{\mu(T_k), \max_{j<k} \mu(T_j) + \mu(p(r_j, r_k))\}$. We will consider two cases. First, we will assume that either $k < P$, or $k = P$ and $\mu(T_k) \leq \max_{j<k} \mu(T_j) + \mu(p(r_j, r_k))$. In that case,

$$\begin{aligned}
\ell &\leq \max_{j<k}(\mu(T_j) + \mu(p(r_j, r_k))) \\
&\leq \max_{j<k}(\ell^* + (1 - \frac{1}{P})\mu(T_j)) \\
&\leq \ell^* + (1 - \frac{1}{P})d\beta \\
&\leq \ell^* + (1 - \frac{1}{P})d\beta\frac{P}{N}\ell^* \\
&= (1 + d(P-1)\frac{\beta}{N})\ell^*.
\end{aligned}$$

Second, we will assume that $k = P$ and $\mu(T_k) > \max_{j<k} \mu(T_j) + \mu(p(r_j, r_k))$. Then

$$\begin{aligned}
\ell &\leq \mu(T_k) \\
&\leq N - (P-1)\beta \\
&\leq \frac{P}{N}(N - (P-1)\beta)\ell^* \\
&= (P - P(P-1)\frac{\beta}{N})\ell^*.
\end{aligned}$$

Hence the length of (σ, π) is at most $\max\{1 + d(P-1)\frac{\beta}{N}, P - P(P-1)\frac{\beta}{N}\}\ell^*$. This bound is as small as possible if $1 + (dP-1)\frac{\beta}{N}$ equals $P - P(P-1)\frac{\beta}{N}$. In that case, $\beta = \frac{1}{d+P}N$. Then ℓ is at most

$$(P - \frac{P^2 - P}{d+P})\ell^* \quad = \quad (1 + \frac{d(P-1)}{d+P})\ell^* \quad = \quad (d + 1 - \frac{d^2 + d}{d+P})\ell^*.$$

Note that for $\beta = \frac{1}{d+P}N$, the number of elements of the β-restricted instance is at most $d + P + 2$ times the number of elements of the original instance. So the number of tasks of the β-restricted instance is polynomial.

Earlier in this section, it was described how a communication-free schedule can be transformed into a feasible schedule. The length of the resulting schedule exceeds that of the communication-free schedule by the total length of the communication operations. Hence we have proved the following theorem.

Theorem 5.9. *Let (T, μ, c, L, o, g, P) be a LogP instance, such that T is a d-ary intree. Let ℓ^* be the length of an optimal schedule for (T, μ, c, L, o, g, P). In polynomial time, a schedule for (T, μ, c, L, o, g, P) of length at most*

$$(d + 1 - \frac{d^2 + d}{d+P})\ell^* + (d(P-1) - 1)(L + o + c_{\max}\max\{o, g\})$$

can be constructed.

5.3 Another approximation algorithm for arbitrary d-ary intrees

In this section, we will construct different decompositions of arbitrary d-ary intrees with arbitrary task lengths. These decompositions consist of forests that are smaller than those constructed by Algorithm d-ARY TREE DECOMPOSITION and consist of more than one tree. The decomposition algorithm can also be used for inforests by assuming that all roots have the same (dummy) parent. The indegree of this dummy root need not be taken into account. The basis of the decomposition algorithm is the following lemma.

Lemma 5.10. *Let (T, μ, c, L, o, g, P) be a β-restricted LogP instance, such that T is an intree and $\mu(T) \geq \beta$. Then T contains a collection of tasks $u_1, \ldots, u_k$, such that $\beta \leq \mu(T(u_1)) + \ldots + \mu(T(u_k)) \leq 2\beta$, $u_1, \ldots, u_k$ have the same parent v and if $k \geq 2$, then v has at least $k+1$ children.*

Proof. This is obvious if T contains only one task. Suppose the lemma holds for all intrees with at most $n-1$ tasks and $n \geq 2$. Let (T, μ, c, L, o, g, P) be a β-restricted instance, such that T is an intree with n tasks and $\beta \leq \mu(T)$. Assume u is the root of T.

Case 1. u has indegree one.
Let v be the child of u. If $\mu(T(v)) \leq \beta - 1$, then $\beta \leq \mu(T(u)) \leq \mu(u) + \mu(T(v)) \leq 2\beta - 1$. Otherwise, $|T(v)| \geq \beta$ and, with induction, $T(v)$ contains a collection of tasks $w_1, \ldots, w_k$ with the same parent x, such that $\beta \leq \mu(T(w_1)) + \ldots + \mu(T(w_k)) \leq 2\beta$ and if $k \geq 2$, then x has at least $k + 1$ children.

Case 2. u has indegree at least two.
We may assume $\mu(T(u)) \geq 2\beta + 1$. Let $v_1, \ldots, v_m$ be the children of u, such that $\mu(T(v_1)) \geq \ldots \geq \mu(T(v_m))$. If $\mu(T(v_1)) \geq \beta$, then, with induction, $T(v_1)$ contains a collection of tasks $w_1, \ldots, w_k$ such that $\beta \leq \mu(T(w_1)) + \ldots + \mu(T(w_k)) \leq 2\beta$, $w_1, \ldots, w_k$ have the same parent x, and if $k \geq 2$, then x has at least $k + 1$ children. So we will assume $\mu(T(v_1)) \leq \beta - 1$. We know that $\mu(T(v_1)) + \ldots + \mu(T(v_m)) = \mu(T(u)) - 1 \geq 2\beta$. Let k be the smallest integer, such that $\mu(T(v_1)) + \ldots + \mu(T(v_k)) \geq \beta$. Then $k \leq m - 1$ and $\mu(T(v_1)) + \ldots + \mu(T(v_k)) \leq \beta - 1 + \mu(T(v_k)) \leq 2\beta - 2$.

□

Like Algorithm d-ARY TREE DECOMPOSITION, Algorithm TREE DECOMPOSITION shown in Figure 8 constructs decompositions of arbitrary intrees by repeatedly removing a subforest.

Algorithm TREE DECOMPOSITION
Input: A β-restricted instance (T, μ, c, L, o, g, P), such that T is an intree with n nodes and an integer k with $1 \leq k \leq n$.
Output: A decomposition $T_1, \ldots, T_{k'}$ of (T, μ, c, L, o, g, P), such that $k' \leq k$ and $\beta \leq \mu(T_i) \leq 2\beta$ for all $i \leq \min\{k - 1, k'\}$.
1. $i := 1$
2. **while** $\mu(T) \geq 2\beta$ **and** $i < k$
3. **do** let $u_{i,1}, \ldots, u_{i,n_i}$ be tasks of T with one parent, such that $\beta \leq \sum_{j=1}^{n_i} \mu(T(u_{i,j})) \leq 2\beta$
4. $T_i := T(u_{i,1}) \cup \ldots \cup T(u_{i,n_i})$
5. $T := T \setminus T_i$
6. $i := i + 1$
7. $T_i := T$

Figure 8: The decomposition algorithm for arbitrary intrees

Let (T, μ, c, L, o, g, P) be a β-restricted instance, such that T is an intree. Let $T_1, \ldots, T_k$ be the decomposition of (T, μ, c, L, o, g, P) constructed by Algorithm TREE DECOMPOSITION using parameter P. Using this decomposition, Algorithm DECOMPOSITION TREE SCHEDULING constructs a communication-free schedule (σ, π) for (T, μ, c, L, o, g, P). If $(T^*, \mu^*, c^*, L, o, g, P)$ is the instance from which (T, μ, c, L, o, g, P) is constructed, then we assume that the topological orders used by Algorithm DECOMPOSITION TREE SCHEDULING are T^*-consistent.

From Lemma 5.1, the length of (σ, π) is at most

$$\max\{\mu(T_k), \max_{i<k} \mu(T_i) + \mu(r_{i,1}, r_{k,1})\},$$

where $r_{i,j}$ is the the root of the j^{th} subtree of T_i.

If $k \leq P - 1$, then $\mu(T_k) \leq 2\beta$. Otherwise, $k = P$ and

$$\mu(T_k) \quad = \quad \mu(T) - (\mu(T_1) + \ldots + \mu(T_{P-1})) \quad \leq \quad \mu(T) - (P - 1)\beta.$$

Define $N = \mu(T)$. Let ℓ be the length of (σ, π) and let ℓ^* be the length of an optimal schedule for (T, μ, c, L, o, g, P). Obviously,

$$\ell^* \quad \geq \quad \frac{1}{P}\mu(T_i) + \mu(p(r_{i,1}, r_{k,1})) \quad \text{and} \quad \ell^* \quad \geq \quad \frac{N}{P}.$$

The length of (σ, π) equals $C(r_{k,1})$, since $r_{k,1}$ is the root of T. Two cases need to be taken into account. First, suppose that either $k < P$, or $k = P$ and $\mu(T_k) \leq \max_{j<k} \mu(T_j) + \mu(p(r_{j,1}, r_{k,1})$. Then

$$\begin{aligned}
\ell &\leq \max_{j<k}(\mu(T_j) + \mu(p(r_{j,1}, r_{k,1}))) \\
&\leq \max_{j<k}(\ell^* + (1 - \frac{1}{P})\mu(T_j)) \\
&\leq \ell^* + 2(1 - \frac{1}{P})\beta \\
&\leq \ell^* + 2(1 - \frac{1}{P})\beta\frac{P}{N}\ell^* \\
&= (1 + 2(P-1)\frac{\beta}{N})\ell^*.
\end{aligned}$$

Second, suppose that $k = P$ and $\mu(T_k) > \max_{j<k} \mu(T_j) + \mu(p(r_{j,1}, r_{k,1}))$. In that case,

$$\begin{aligned}
\ell &\leq \mu(T_k) \\
&\leq N - (P-1)\beta \\
&\leq \frac{P}{N}(N - (P-1)\beta)\ell^* \\
&= (P - P(P-1)\frac{\beta}{N})\ell^*.
\end{aligned}$$

Hence the length of (σ, π) is at most $\max\{1 + 2(P-1)\frac{\beta}{N}, P - P(P-1)\frac{\beta}{N}\}\ell^*$. This bound is as small as possible if $1 + (2P-1)\frac{\beta}{N}$ and $P - P(P-1)\frac{\beta}{N}$ are equal. In that case, $\beta = \frac{1}{P+2}N$ and ℓ is at most

$$(1 + 2\,\frac{P-1}{P+2})\ell^* \quad = \quad (3 - \frac{6}{P+2})\ell^*.$$

Suppose (T, μ, c, L, o, g, P) is constructed from original instance $(T^*, \mu^*, c^*, L, o, g, P)$. Similar to Lemma 5.7, we can prove that the subtasks in T of a task u of T^* are scheduled on the same processor. There may be other tasks that are scheduled between these subtasks, but it is not difficult to transform (σ, π) into a communication-free schedule (σ^*, π^*) for $(T^*, \mu^*, c^*, L, o, g, P)$ with the same length. The number of communicating pairs does not increase, because no task needs to be scheduled on a different processor. So the communication requirements of both communication-free schedules are equal.

Each decomposition forest T_i consists of a collection of trees whose roots have the same parent. Using Lemma 5.10, if a decomposition forest consists of more than one tree, then the parent of the roots of these trees has another child. In addition, decomposition forest T_k consists of one tree. Hence the number of roots of the decomposition forests is bounded by $\max\{d-1, 1\}(P-1)+1 = (d-1)(P-1)+1$, where d is the maximum indegree in T.

Since the indegree of a possible dummy root need not be considered, we have proved the following theorem.

Theorem 5.11. *Let (T, μ, c, L, o, g, P) be a LogP instance, such that T is a d-ary inforest. Let ℓ^* be the length of an optimal schedule for (T, μ, c, L, o, g, P). In polynomial time, a schedule for (T, μ, c, L, o, g, P) of length at most*

$$(3 - \frac{6}{P+2})\ell^* + (d(d-1)(P-1) - 1)(L + o + c_{\max}\max\{o, g\})$$

can be constructed.

In Sections 5.2 and 5.3, two algorithms are presented that construct decomposition of arbitrary d-ary intrees with arbitrary task lengths. The first constructs decompositions consisting of subforests that consist of one intree. The sizes of these subforests differ by at most a factor of d. The second constructs decompositions into a collection of subforests that consist of at most $d-1$ intrees. The sizes of these subforests differ at most a factor of two.

The length of the schedules constructed by Algorithm DECOMPOSITION TREE SCHEDULING using the decompositions constructed by Algorithms d-ARY TREE DECOMPOSITION and TREE DECOMPOSITION consist of two parts; the first depends on the task lengths and the precedence constraints and is independent of the communication requirements, the second only depends only on the communication requirements.

The decompositions of Algorithms d-ARY TREE DECOMPOSITION and TREE DECOMPOSITION give a trade-off between computation and communication: the lengths of the schedules constructed using decompositions constructed by Algorithm d-ARY TREE DECOMPOSITION have a rather large computation part and a small communication part, whereas the lengths the schedules constructed using decompositions of Algorithm TREE DECOMPOSITION have a small computation part and a larger communication part.

Concluding remarks

The algorithm presented in Section 5 can be used in a very general setting. The decompositions are used by Algorithm DECOMPOSITION TREE SCHEDULING to construct communication-free schedules. These schedules can be transformed into feasible schedule for any model of parallel computation by introducing communication tasks between every communicating pair.

Hence, for any model of parallel computation, we can construct schedules on P processors for full d-ary intrees with unit-length tasks whose lengths are at most $2 + \frac{2}{d}$ times that of an optimal schedule plus the duration of $(d+1)P - 1$ communication operations.

In addition, for arbitrary d-ary intrees, schedules of length at most $d + 1 - \frac{d^2+d}{d+P}$ times the length of a minimum-length schedule plus the duration of $d(P-1) - 1$ communication operations can be constructed in polynomial time. The same holds for schedules of length at most $3 - \frac{6}{P+2}$ times the length of an optimal schedule plus the duration of $d(d-1)(P-1) - 1$ communication operations.

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 in PRAMs. *Theoretical Computer Science*, 71:3–28, 1990.

[4] P. Chrétienne. Tree scheduling with communication delays. *Discrete Applied Mathematics*, 49:129–141, 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*, pages 65–90. John Wiley & Sons, 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] 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.

[8] D.E. Culler, A.C. Dusseau, R.P. Martin and K.E. Schauser. Fast parallel sorting under LogP: from theory to practice. In T. Hey and J. Ferrante, editors, *Portability and Performance for Parallel Processing*, chapter 4, pages 71–98. John Wiley & Sons, New York, 1994.

[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] 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.

[11] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

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

[13] 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.

[14] 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.

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

[16] T.-S. Hsu and D.R. Lopez. Bounds and algorithms for a practical task allocation model. In T. Asano, Y. Igarashi, H. Nagamochi, S. Miyano and S. Suri, editors, *Proceedings of the 7th International Symposium on Algorithms and Computation*, volume 1178 of *Lecture Notes in Computer Science*, pages 397–406, Berlin, 1996. Springer-Verlag.

[17] 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.

[18] S.R. Kosaraju. Parallel evaluation of division-free arithmetic expressions. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 231–239, 1986.

[19] M. Kunde. Nonpreemptive LP-scheduling on homogeneous multiprocessor systems. *SIAM Journal on Computing*, 10(1):151–173, February 1981.

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

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

[23] 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*, volume 854 of *Lecture Notes in Computer Science*, pages 277–288, Berlin, 1994. Springer-Verlag.