II Scheduling in the LogP model
8 The LogP model

Part II is concerned with scheduling in the LogP model. In this chapter, the LogP model is presented as a scheduling model. In Section 8.1, the communication requirements of the LogP model are presented. The general problem instances for LogP scheduling are introduced in Section 8.2, feasible schedules for such instances are presented in Section 8.3. In Section 8.4, previous results concerning scheduling in the LogP model are presented. An outline of the second part of this thesis is presented in Section 8.5.

8.1 Communication requirements

The LogP model [21] is a model of a distributed memory computer. It consists of a number of identical processors connected by a communication network. Each processor has an unlimited amount of local memory. The processors execute a computer program in an asynchronous manner: one processor can execute a task while another is involved in a communication action. Communication is modelled by message-passing: data is transferred between the processors by sending messages through the communication network.

The LogP model captures the characteristics of a real parallel computer 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. [21] make the following assumptions. The communication network is assumed to be of finite capacity: at each time at most $\lfloor \frac{L}{g} \rfloor$ 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 $\lfloor \frac{L}{g} \rfloor$ 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.

We will consider a common data semantics [25]: the children of a task $u$ all need the complete result of $u$. So the result of the execution of a task needs to be sent at most once to any other processor even if a processor executes more than one child of $u$. 

103
The communication between processors in the LogP model works as follows. Consider two different processors $p_1$ and $p_2$. Assume processor $p_1$ executes a task $u_1$ and the processor $p_2$ a child $u_2$ of $u_1$. Then the result of the execution of $u_1$ must be transferred from processor $p_1$ to processor $p_2$ before $u_2$ can be executed. Assume the result of $u_1$ is contained in two messages. Then two messages must be sent from processor $p_1$ to processor $p_2$. Figure 8.1 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 8.1. Communication between two processors in the LogP model](image)

The first message can be sent by processor $p_1$ immediately after the completion of $u_1$. After this message has been submitted to the communication network, exactly $L$ time units are used to send it to processor $p_2$ through the network. Then 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 the starting times of the receive operations differ at least $g$ time units. After the second message has been received by processor $p_2$, $u_2$ can be scheduled.

If another child of $u_1$ is scheduled after $u_2$ on processor $p_2$, then no additional communication is necessary: this child can be executed immediately after $u_2$. This is due to the fact that the result of $u_1$ has already been transferred from processor $p_1$ to processor $p_2$.

Under a common data semantics [25], the children of a task $u$ all need the complete result of $u$ and the result of a task has to be sent to any processor at most once. Under an independent data semantics [25], each child of a task $u$ needs a separate part of the result of $u$. Using an independent data semantics, a separate set of messages has to be sent for every child of $u$ that is not scheduled on the same processor as $u$. Note that if every task has at most one child, then there is no difference between a common data semantics and an independent data semantics: if a task $u$ has exactly one child, then it requires the complete result of $u$. In addition, the problem of scheduling outforests under an independent data semantics is the same as scheduling inforests (under either an independent data semantics or a common data semantics).

### 8.2 Problem instances

The general scheduling instances introduced in Chapter 2 have to be extended to obtain LogP scheduling instances. These instances are extended with the parameters of the LogP model and
the sizes of the results of the tasks. Hence we will consider instances \((G, \mu, c, L, o, g, P)\), such that tuple \((G, \mu, c)\) describes a computer program and \((L, o, g, P)\) contains the parameters of the LogP model. In a tuple \((G, \mu, c, L, o, g, P)\), \(G\) is a precedence graph, \(\mu : V(G) \rightarrow \mathbb{Z}^+\) is a function that assigns an execution length to every task of \(G\) and \(c : V(G) \rightarrow \mathbb{N}\) is a function that specifies the number of messages needed to send the result of a task of \(G\) to another processor. Because the result of a sink of \(G\) is not sent to any processor, we will assume that \(c(u)\) equals zero for all sinks \(u\) of \(G\). In the remainder of Part II, we will only consider instances \((G, \mu, c, L, o, g, P)\), such that \(c(u) \geq 1\) for every task \(u\) of \(G\) that is not a sink of \(G\). All algorithms presented in the following chapters can be easily generalised to scheduling instances \((G, \mu, c, L, o, g, P)\) with arbitrary functions \(c\).

Like for scheduling in the UCT model, some special instances will be considered. If all tasks have unit length, then \(\mu\) will be omitted. In addition, if \(c(u)\) equals one for all tasks \(u\) of \(G\) with outdegree at least one, then \(c\) will be left out. So the instance \((G, L, o, g, P)\) corresponds to the instance \((G, \mu, c, L, o, g, P)\), such that \(\mu(u) = 1\) for all tasks \(u\) of \(G\) and \(c(u) = 1\) for all tasks \(u\) of \(G\) with outdegree at least one and \(c(u) = 0\) for all sinks \(u\) of \(G\).

### 8.3 Feasible schedules

In the LogP model, processors communicate by sending messages to each other. For each task \(u\), messages have to be sent to all processors that execute a child of \(u\) except the processor that executes \(u\). So the corresponding send and receive operations may be scheduled for all processors but one. Since we assume a common data semantics, no message needs to be sent to the same processor twice.

Consider a task \(u_1\) and one of its children \(u_2\) that are scheduled on different processors. Assume \(u_1\) is executed on processor \(p_1\) and \(u_2\) on processor \(p_2 \neq p_1\). Then \(c(u_1)\) messages \(m_{u_1,1}, \ldots, m_{u_1,c(u)}\) 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 \(G\) that is not a sink 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 \(G\) that is not a sink 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 \(\mu(u) = o\).

Note that the communication operations have length zero if \(o\) equals zero. Because there must be a delay of at least \(g\) time units between the starting times of two consecutive send operations or two consecutive receive operations on the same processor, the presence of zero-length communication operations is not the same as the absence of communication operations.

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

**Definition 8.3.1.** A schedule \((\sigma, \pi)\) for \((G, \mu, c, L, o, g, P)\) is called 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 \( V(G) \cup C(G, P, c) \), 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 \( i \leq c(u_1) \),
   \[
   \sigma(s_{u_1, \pi(u_2), i}) = \pi(u_1), \quad \sigma(r_{u_1, \pi(u_2), i}) = \pi(u_2), \quad \sigma(s_{u_1, \pi(u_2), i}) + \sigma(r_{u_1, \pi(u_2), i}) = \sigma(s_{u_1, \pi(u_2), i}) + o + L \quad \text{and} \quad \sigma(u_2) \leq \sigma(r_{u_1, \pi(u_2), i}) + o;
   \]
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 children 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 first constraint states that all tasks of \( G \) have to be executed. The second and third correspond to the constraints for feasible communication-free schedules: a processor cannot execute two tasks at the same time and a task must 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 [21], processors can send messages to other processors, unless the number of messages in transit from or to one processor exceeds \( \left\lceil \frac{L}{g} \right\rceil \), in which case the sending 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 \( \left\lfloor \frac{L + \max(o, g) - 1}{\max(o, g)} \right\rfloor + 1 \leq \left\lceil \frac{L}{g} \right\rceil \). So we do not need to consider stalling.

Constructing a schedule for an instance \((G, \mu, c, L, o, g, P)\) 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 \((G, \mu, c, L, o, g, P)\) uses at
least $\Theta(\sum_{u \in V(G)} c(u))$ time. If $c_{\text{max}} = \max_{u \in V(G)} c(u)$ is not bounded by a polynomial in $n$ and $\log \max_{u \in V(G)} \mu(u)$, 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 $c_{\text{max}}$ is not exponentially large. In the rest of Part II, we do not want to focus on the time needed to schedule the communication operations. Hence we will assume that $c_{\text{max}}$ is bounded by a constant. However, the time complexity of the algorithms presented in the remaining chapters of Part II remains polynomial if $c_{\text{max}}$ is bounded by a polynomial in $n$ and $\log \max_{u \in V(G)} \mu(u)$: the time complexity of the algorithms must be increased by $O(nc_{\text{max}})$ to account for the assignment of a starting time and a processor to each communication operation.

This section will be concluded with two examples of feasible schedules. The first is a schedule for the same graph as the one in Sections 2.1 and 3.4.

![Figure 8.2. An instance $(G, \mu, 1, 1, 1, 2)$](image)

![Figure 8.3. A feasible schedule for $(G, \mu, 1, 1, 1, 2)$](image)

**Example 8.3.2.** Consider the instance $(G, \mu, 1, 1, 1, 2)$ shown in Figure 8.2. Each task of $G$ is labelled with its name, its execution length and the number of messages required to send its result to another processor. The instance $(G, \mu, 1, 1, 1, 2)$ corresponds to the general scheduling instance $(G, \mu, 2)$ shown in Figure 2.1 and the UCT instance $(G, \mu, 2, D)$ shown in Figure 3.1. A feasible schedule for $(G, \mu, 1, 1, 1, 2)$ is shown in Figure 8.3. $a_1$ and $a_2$ are scheduled on different processors. $b_2$ is a common child of $a_1$ and $a_2$. So the result of $a_1$ is sent to the second processor. This is represented by tasks $s_{a_1}$ and $r_{a_1}$. Note that there is a delay of one time unit between the completion time of $s_{a_1}$ and the starting time of $r_{a_1}$. Since $a_1$ is the only parent of $b_1$ and $b_2$ is
the only parent of \( c_2 \), these tasks can be scheduled without extra communication on the first and second processor, respectively. \( c_1 \) is a child of \( b_1 \) and \( b_2 \). Because its parents are scheduled on different processors, the result of \( b_2 \) is sent to the first processor before \( c_1 \) is executed. Similarly, the result of \( c_2 \) is sent to the first processor before \( d_1 \) starts.

The next example shows a schedule for an instance \((G, \mu, c, L, o, g, P)\) in which \( g \) exceeds \( o \). It shows that the idle time between consecutive communication operations can be used to execute tasks.

**Figure 8.4.** An instance \((G, \mu, c, 2, 1, 2, 2)\)

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>s_x,1</td>
<td>y_1</td>
<td>s_x,2</td>
<td>y_2</td>
<td>s_x,3</td>
<td>y_5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_x,1</td>
<td>r_x,2</td>
<td>r_x,3</td>
<td>y_3</td>
<td>y_4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 8.5.** A feasible schedule for \((G, \mu, c, 2, 1, 2, 2)\)

**Example 8.3.3.** Consider the instance \((G, \mu, c, 2, 1, 2, 2)\) shown in Figure 8.4. It is not difficult to see that the schedule shown in Figure 8.5 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. This is due to the fact that we assume a common data semantics: the complete result of \( x \) is sent to the second processor and it has to be sent to this processor exactly once. Under an independent data semantics, two separate sets of messages must be sent to the second processor: a set of messages for \( y_3 \) and one for \( y_4 \).

Examples 8.3.2 and 8.3.3 show that schedules in the LogP model are very different from communication-free schedules and from schedules in the UCT model. However, communication-free scheduling and scheduling in the UCT model can be seen as special cases of scheduling in the LogP model: if all tasks have unit length or the number of processors is unrestricted, then any communication-free schedule can be viewed as a schedule in the LogP model with parameters \( L = o = g = 0 \) and any schedule in the UCT model as a schedule in the LogP model with parameters \( L = 1 \) and \( o = g = 0 \).
A feasible schedule $(\sigma, \pi)$ for an instance $(G, m, D)$ in the UCT model can be transformed into a feasible schedule for the instance $(G, c, 1, 0, 0, m)$ in the LogP model by scheduling the send and receive operations. For all tasks $u$ of $G$, all processors $p \neq \pi(u)$ that execute a child of $u$ and all $i \in \{1, \ldots, c(u)\}$, send operation $s_{u,p,i}$ must be executed at time $\sigma(u) + 1$ on processor $\pi(u)$ and receive operation $r_{u,p,i}$ at time $\sigma(u) + 2$ on processor $p$. Since $g = o = 0$, the resulting schedule is a feasible schedule for $(G, c, 1, 0, 0, m)$. A feasible communication-free schedule for an instance $(G, \mu, m)$, such that $\mu(u) = 1$ for all tasks $u$ of $G$, can be transformed into a feasible schedule for the instance $(G, c, 0, 0, 0, m)$ in the LogP model in a similar way. Moreover, communication-free schedules for instances $(G, \mu, \infty)$ can be transformed into feasible schedules for instances $(G, \mu, c, 0, 0, 0)$ and schedules in the UCT model for instances $(G, \mu, \infty, D)$ into feasible schedules for instances $(G, \mu, c, 1, 0, 0)$. Both transformations do not change the starting time of any tasks, but they may schedule tasks on different processors.

8.4 Previous results

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, 24], broadcast [54] and the Fast Fourier Transform [20].

In addition, Löwe and Zimmermann [63, 95] 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. [64] proved the same result for a generalisation of the LogP model. Moreover, Löwe and Zimmermann [63] presented an algorithm that constructs schedules of length at most twice as long as a minimum-length schedule plus the duration of the sequential communication operations.

Simultaneously to my research on scheduling in the LogP model, Kort and Trystram [55] studied the problem of scheduling in the LogP model. They presented three algorithms for scheduling send graphs under an independent data semantics [25]. They proved that if $g$ equals $o$ and all sinks or all messages have the same length, then a minimum-length schedule for a send graph on an unrestricted number of processors can be constructed in polynomial time. Because scheduling send graphs under an independent data semantics corresponds to scheduling receive graphs (under a common data semantics), their result also shows that minimum-length schedules for receive graphs on an unrestricted number of processors can be constructed in polynomial time if $g$ equals $o$ and all sources have the same execution length or all message lengths are equal. In addition, Kort and Trystram [55] showed that if all sinks have the same length and this length is at least $\max\{g, 2o + L\}$, then a minimum-length schedule for a send graph on two processors can be constructed in linear time.

8.5 Outline of the second part

The remaining chapters of Part II are concerned with the problem of constructing minimum-length schedules in the LogP model. In the next chapter, we study the problem of scheduling
send graphs. It is proved that constructing minimum-length schedules for a send graph on an unrestricted number of processors is a strongly NP-hard optimisation problem. A polynomial-time algorithm is presented that constructs schedules for send 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 send graph on $P$ processors can be constructed in polynomial time.

In Chapter 10, two polynomial-time approximation algorithms for scheduling receive graphs are presented. The first is a 3-approximation algorithm for scheduling receive graphs on an unrestricted number of processors. For each constant $k \in \mathbb{Z}^+$, the second algorithm can construct schedules for receive graphs on $P$ processors that are at most $3 + \frac{1}{k+1}$ times as long as minimum-length schedules on $P$ processors. Moreover, it is proved that if all task lengths are equal, then a minimum-length schedule for a receive graph on an unrestricted number of processors can be constructed in polynomial time.

In Chapter 11, two algorithms are presented that decompose inforests into subforests whose sizes do not differ much. Using the decompositions constructed by the first algorithm, schedules for $d$-ary inforests on $P$ processors are constructed that have a length that is at most the sum of $d + 1 - \frac{d^2 + d}{d+1}$ times the length of a minimum-length schedule on $P$ processors and the duration of $d(P - 1) - 1$ communication actions. The decompositions constructed by the other algorithm can be used to construct schedules on $P$ processors with a length that is at most the sum of $3 - \frac{6}{P-2}$ times the length of a minimum-length schedule on $P$ processors and the duration of $d(d - 1)(P - 1) - 1$ communication actions.