Scheduling of Parallel Programs in Single-Bus Multiprocessor Systems

Lucian Finta, Zhen Liu

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Scheduling of Parallel Programs in Single-Bus Multiprocessor Systems

Lucian FINTA Zhen LIU

N° 2302

Mai 1994

PROGRAMME 1

Architectures parallèles,

bases de données,

réseaux et systèmes distribués

ISSN 0249-6399

INRIA
SOPHIA ANTIPOLIS

Scheduling of Parallel Programs in Single-Bus Multiprocessor Systems*

Lucian FINTA Zhen LIU

Programme 1 — Architectures parallèles, bases de données, réseaux
et systèmes distribués
Projet MISTRAL

Rapport de recherche n°2302 — Mai 1994 — 42 pages

Abstract: Consider a scheduling problem of parallel computations in multiprocessor systems. Let a parallel program be represented by a task graph, where vertices represent tasks and arcs represent the communications between the tasks. An interprocessor communication time incurs when two tasks assigned to two different processors have to communicate. Such a scheduling problem has recently been studied in the literature, mostly for the case where interprocessor communication times are fully determined. In this paper, we consider the scheduling problem with resource constraints. More specifically, we consider the case where all interprocessor communications take place on a single bus. We show that even for very specific subproblems, the minimization of the makespan of parallel programs in such a single-bus multiprocessor system is NP-hard. Thus, the general scheduling problem of parallel programs with communication resource constraints is NP-hard. We consider several variants of the problem: tasks with or without preallocation, communications with independent-data semantics or common-data semantics. Our results are extended to the cases of blocking communications and of broadcasting communications, and can be applied to multiprocessor systems with shared memory.

Key-words: Scheduling, Makespan, Precedence Constraint, Communication bus, Resource Constraint, NP-completeness.

(Résumé : tsvp)

*__Correspondence__: Zhen LIU, INRIA, Centre Sophia Antipolis, 2004 route des Lucioles, B.P. 93, 06902 Sophia-Antipolis, France. e-mail: liu@sophia.inria.fr

Unité de recherche INRIA Sophia-Antipolis
2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex (France)
Téléphone : (33) 93 65 77 77 – Télécopie : (33) 93 65 77 65

Ordonnancement de programmes parallèles sur des machines multiprocesseurs à bus unique

Résumé : Nous considérons un problème d'ordonnancement pour les calculs parallèles dans un système multiprocesseur. Un programme parallèle est représenté par un graphe de tâches, où les sommets représentent les tâches et les arcs les communications entre les tâches. Un tel problème a été récemment étudié dans la littérature dans le cas où les temps de communication ne dépendent pas de l'état du système. Dans cet article, nous considérons le problème d'ordonnancement avec contrainte de ressources. Plus précisement, nous étudions le cas où toutes les communications ont lieu sur un bus unique. Nous prouvons que, même pour des sous-problèmes spécifiques, la minimisation de la durée d'ordonnancement pour les programmes parallèles dans un tel système multiprocesseur est NP-difficile. Cela implique que le problème général d'ordonnancement pour les programmes parallèles avec des contraintes des ressources de communication est NP-difficile. Nous analysons plusieurs variantes du problème : les tâches avec ou sans allocation au préalable, les communications avec sémantique de données indépendantes ou sémantique de données communes. Nos résultats s'étendent aux cas des communications avec blocage et des communications avec diffusion, et peuvent s'appliquer aux systèmes multiprocesseurs à mémoire partagée.

Mots-clé : ordonnancement, durée d'ordonnancement, contraintes de précédence, bus de communication, contraintes de ressources, complexité, NP-complétude.

1 Introduction

Scheduling of parallel programs in multiprocessor systems is one of the important issues in parallel computing. Parallel programs are usually represented by task graphs, where vertices represent tasks and arcs the communications between the tasks. An interprocessor communication time incurs when two tasks assigned to two different processors have to communicate. A task can start execution only when all its predecessor tasks have completed execution and the communications between its predecessor tasks and the task have taken place. Our goal is to minimize the makespan of a program, i.e. the maximum task completion time.

Such a scheduling problem has been receiving growing interest in the literature recently. Most of the studies have been focused on the case where interprocessor communication times are fully determined, i.e., interprocessor communication times are constants which are possibly dependent of the message length and the distance between processors, but are otherwise independent of the status of system (in particular, the communication network).

Rayward-Smith [11] first analyzed the case where tasks have unit-execution-time (UET) and interprocessor communications have unit-communication-time (UCT). He showed that the minimization of makespan is NP-hard. Picouleau [9] and Hoogeveen et al. [5] analyzed the complexity (polynomial or NP-complete) of the decision problem whether the minimum makespan of a program can be smaller than a threshold. Various heuristics were proposed in [4, 8, 6], and a general worst-case analysis of greedy heuristics was given in Liu [7]. There are also studies on the case of infinite number of processors, see e.g. Chretienne [1] and Picouleau [10]. The reader is referred to Veltman et al. [13] for a classification of the variants of the scheduling problem, and to Chretienne and Picouleau [2] for a survey on the complexity results, optimal and approximate algorithms.

In this paper, we consider the scheduling problem with resource constraints. More specifically, we consider the case where all interprocessor communications take place on a single bus. We show that even for very specific subproblems, the minimization of the makespan of parallel programs in such a single-bus multiprocessor system is NP-hard. Thus, the general scheduling problem of parallel programs with communication resource constraints is NP-hard.

We consider several variants of the problem. Tasks may or may not be preallocated to specific processors. Communications between tasks can have independent-data or common-data semantics (see definitions below). Inter-processor communications can have blocking or nonblocking mechanisms.

The paper is organized as follows. In Section 2, we describe the scheduling problem and its variants in detail. We will first consider nonblocking communications. In Section 3 we prove the NP-hardness of problems when communications between tasks have independent-data semantics. In Section 4 we consider the case of communications with common-data semantics. In Section 5, we extend these results to the cases of blocking communications and of broadcasting communications. Finally, in Section 6, we provide some concluding remarks.

2 Problem Description

The multiprocessor system under consideration contains m parallel processors connected to a single communication bus, where m is an arbitrary positive integer. All communications between processors occur on the bus. At any time, only one processor can send a message over the bus.

Two *communication mechanisms* will be considered. The first one is *one-to-one communication*, where only one of the processors can be the receiver of a message. The second one is *broadcasting communication*, where several processors can be the receivers of a message, provided they are the destinations.

When a processor is sending or receiving a message, it can stop processing a task until the communication is finished. This communication mechanism is called communication with *blocking*. In what follows we shall first consider nonblocking communications. Thus, a processor can execute tasks while sending messages. The case of blocking communications will be considered in Section 5.

A parallel program is represented by a directed acyclic graph $G = (V, E)$, referred to as *task graph*, where vertices in V represent tasks of the parallel program, arcs of E represent data dependence or communication relations between tasks. If $(i, j) \in E$, then task j has to wait for results of task i before starting its own execution. Denote by $P(i)$ and $S(i)$ the sets of immediate predecessors and successors of task $i \in V$,

i.e. $P(i) = \{v : (v,i) \in E\}$, $S(i) = \{v : (i,v) \in E\}$. Task i is an *initial task* (resp. a *final task*) if $P(i) = \emptyset$ (resp. $S(i) = \emptyset$).

Task running times on the processors are assumed to be known constants. In most of the studies in the paper, we consider the case where tasks have UET.

The time for data transfer between tasks allocated to the same processor is assumed to be negligible. However, data transfer between two tasks allocated to different processors are carried out through message passing. The set of data to be transferred from one task to another can be specified as a weight of an arc in the task graph. In general, as discussed in Veltman et al. [13], if two immediate successors u, v of task i are assigned to the same processor and if task u starts execution prior to task v, then only those data that are useful to v but not to u need to be transferred from task i to task v. In order to simplifying discussions, we shall consider two extreme cases of the *communication semantics*: *independent data* and *common data*. In the case of independent-data communication semantics, the sets of data to be transferred from a task to its immediate successors are assumed to be different. In the case of common-data communication semantics, however, the sets of data to be transferred from a task to its immediate successors are assumed to be the same. Only in this case one can use broadcasting communication mechanisms.

In both cases, we can simply specify the communications between tasks by the amount of data to be transferred, or even simpler, the communication time for the data transfer if the communication takes place on the bus. In this paper, we assume that these communication times are specified (e.g. as the weights of arcs in the task graph). We will consider the simplest case, i.e. UCT, where all communications on the bus take unit time.

Note that in the literature, most of the works (see e.g. [2]) on scheduling of parallel programs with communication are on the independent-data communication semantics.

A *schedule* of the parallel program in the multiprocessor system defines for each task the processor the task is assigned to, and the time epoch when the task starts execution. The schedule defines also for each communication the time slot when the message is sent over the bus. The schedule is feasible if at any time only one task is running on a processor and only one communication occurs on the bus, and if a task starts its execution on a processor only after all data needed from its predecessors

are ready on the processor. No *task duplication* is allowed, i.e., a task can be run only on one processor.

Each task may have a subset of accessible processors, i.e. processors to which the task can be assigned to. We shall analyze two extreme cases. In the first case, all sets of accessible processors are singletons. This case is referred to as *tasks with preallocation*. We will further assume that the tasks assigned to the same processor run on the processor according to a predefined (total) order. In this case, the only scheduling decisions to be made is the schedule of communications on the bus. In the second case, all sets of accessible processors are identical to the set of processors in the system. This case is referred to as *tasks without preallocation*. In this case, the scheduling decisions concern both task assignment/scheduling on processors and communication scheduling on the bus.

Given a feasible schedule σ, let $C_i(\sigma)$ denote the completion time of the task i in the schedule σ. The goal of our study is to minimize the length of the schedule, or the *makespan*, i.e. the maximum of task completion times: $C_{\max}(\sigma) = \max_{i \in V} C_i(\sigma)$.

In this paper we obtain "negative" results of the scheduling problem, i.e., we prove the NP-hardness of the makespan minimization problems. In order to do this, we consider the associated *decision problems*:

(PR1): Given a task graph $G = (V, E)$ with UET-UCT and independent-data communication semantics, m processors, and a time limit T, is there a feasible schedule σ such that $C_{\max}(\sigma) \leq T$?

(PR2): Given a task graph $G = (V, E)$ with UET-UCT and independent-data communication semantics, m processors, task allocations, and a time limit T, is there a feasible schedule σ such that $C_{\max}(\sigma) \leq T$?

(PR3): Given a task graph $G = (V, E)$ with UET-UCT and common-data communication semantics, m processors, and a time limit T, is there a feasible schedule σ such that $C_{\max}(\sigma) \leq T$?

(PR4): Given a task graph $G = (V, E)$ with UET-UCT and common-data communication semantics, m processors, task allocations, and a time limit T, is there a feasible schedule σ such that $C_{\max}(\sigma) \leq T$?

We will show that these decision problems are NP-complete so that the corresponding optimization problems are NP-hard.

3 Scheduling with Independent-Data Communication Semantics

3.1 Scheduling without Task Preallocation

Consider first the case where tasks have no preallocations. We prove that the problem is NP-hard even for four processors with UET, and for two processors with two different units of task execution times.

For the case of four processors, we polynomially transform the well-known *partition* problem to (PR1). Recall the definition of partition problem, where $I\!N_+ = \{1, 2, \ldots\}$.

> *Partition problem*: Given a finite set A of items and a size $s(a) \in I\!N_+$ for each $a \in A$, is there a subset $A' \subseteq A$ and some $B \in I\!N_+$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a) = B$?

Lemma 1 *The partition problem polynomially transforms to (PR1) with UET and four processors.*

Proof. Given an instance of the partition problem as above, we construct the following instance of (PR1) with UET and four processors, such that there exists a feasible schedule σ if and only if the partition problem has a solution.

Let $A = \{a_1, \cdots, a_n\}$ be the set of items in the partition problem. The time limit for the scheduling function is $T = 2B + 4$. The task graph $G = (V, E)$ is constructed as follows.

- There are two chains, referred to as x-chain and y-chain, of T and $T - 2$ tasks with UET, denoted by $x_1 \rightarrow x_2 \rightarrow \cdots \rightarrow x_T$ and $y_3 \rightarrow y_4 \rightarrow \cdots \rightarrow y_T$, where the symbol "$\rightarrow$" represents an arc.
- The first task (x_1) in the x-chain is the immediate predecessor of the first task (y_3) in the y-chain. For $n + 2 \leq i \leq T - 2$, there is an arc from task x_i to task y_{i+2}.

- For each a_i, $1 \leq i \leq n$, there is a chain of $2s(a_i)$ tasks with UET, denoted by $z_{i,1} \rightarrow z_{i,2} \rightarrow \cdots \rightarrow z_{i,2s(a_i)}$, tasks z_{i1} being immediate successors of task x_1 (first task in the x-chain).

The total number of tasks in the graph is $|V| = 4T - 10$. Figure 1 illustrates the task graph that we construct for each instance of the partition problem.

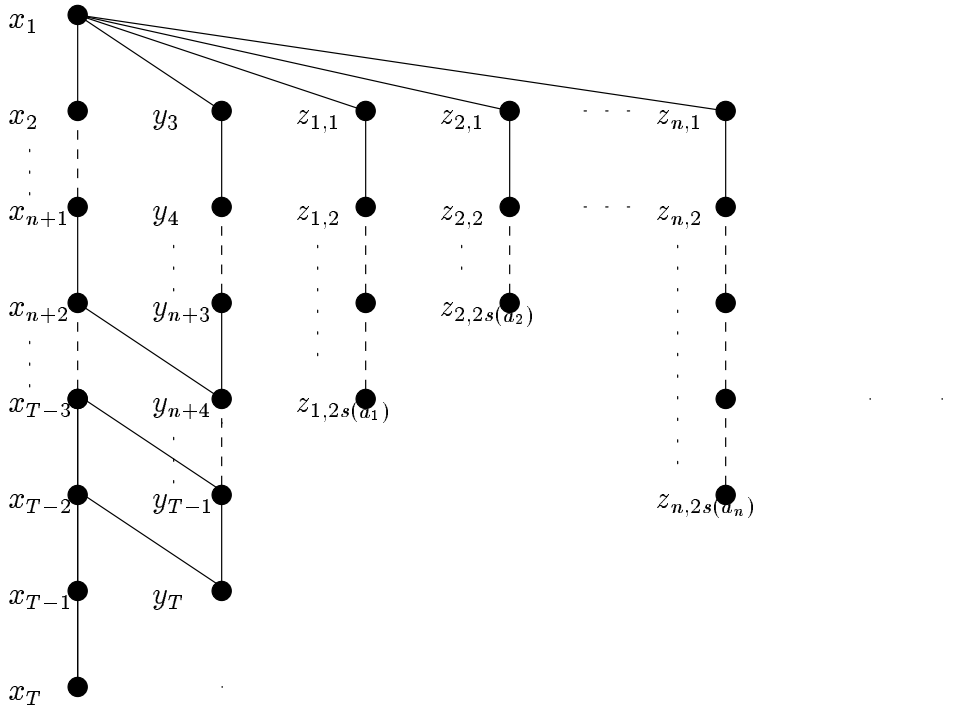


Figure 1: Task graph instance of (PR1) corresponding to the instance of partition problem.

Since x-chain has T tasks, there should be no interprocessor communication between tasks of x-chain in order to have a feasible schedule with length T (note that it is impossible to have a feasible schedule with length strictly smaller than T). Thus, these T tasks must be assigned to the same processor, say processor 1.

Note that no other tasks can be executed on processor 1 in any feasible schedule of length T, all successors of x_1 (except for tasks of x-chain) must be processed

on other processors, so that all outgoing arcs from task x_1 (except for $x_1 \rightarrow x_2$) correspond to an interprocessor communication.

Let x_1 starts execution at time 0. Then task y_3 can start at earliest at time 3. Since y-chain has $T-2$ tasks, y_3 should start at least at time 3 in order to have a feasible schedule with length T. Therefore, the first communication on the bus corresponds to $x_1 \rightarrow y_3$. Moreover, there should be no interprocessor communication between tasks of y-chain. Thus, these $T-2$ tasks must be assigned to the same processor, say processor 2, and no other tasks can be assigned to it.

It then follows that communications corresponding to the arcs $x_i \rightarrow y_{i+2}, n+2 \leq i \leq T-2$, are critical in the sense that the bus has to send message $x_i \rightarrow y_{i+2}$ as soon as task x_i completes. Therefore, the bus is occupied by communications between processors 1 and 2 in the time interval $[n+2, T-1)$. Recall that there is a communication between processors 1 and 2 in the time interval $[1, 2)$.

Since a communication on the bus can only be between two tasks on two different processors, the bus can be occupied for communication only in the time interval $[1, T-1)$. Therefore, the bus can take care of at most n communications (in the time interval $[2, n+2)$) in any feasible schedule. Hence, only communications $x_1 \rightarrow z_{i1}$, $i = 1, 2, \ldots, n$, can be dealt with by the bus. In other words, there should be no interprocessor communications between processors 3 and 4 which execute tasks of z-chains.

It then follows that for any feasible schedule, the task assignment should be such that the z-chains are not broken, i.e. tasks belonging to the same z-chain are assigned to the same processor.

Due to the facts that processors 3 and 4 can start executing tasks of z-chains only after times 3 and 4 (because of communications $x_1 \rightarrow z_{i1}$), respectively, and that the lengths of z-chains are even numbers which sum to $2T-8 = 2B$, a feasible schedule exists if and only if there is a subset $A' \subset A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A-A'} s(a) = B$. Note that since the lengths of z-chains are at least 2, it is possible for the bus to deal with the communications $x_1 \rightarrow z_{i1}$ successively without delaying executions of z-chain tasks after time epoch 4. ∎

As a consequence of Lemma 1, we have

Theorem 1 *The decision problem (PR1) with 4 processors is NP-complete.*

Corollary 1 *In a multiprocessor system with a single bus, the makespan minimization of a task graph with UET-UCT and independent-data communication semantics is NP-hard, even if there are only 4 processors.*

The above corollary implies that the makespan minimization of a task graph with UET-UCT and independent-data communication semantics is NP-hard in general. We can also show that when tasks have either one or two units of execution times, the makespan minimization is NP-hard even if there are only 2 processors.

Theorem 2 *In a two-processor system with a single bus, the makespan minimization of a task graph with UCT is NP-hard if tasks can have either one or two units of execution times.*

Proof. The proof is analogous to that of Theorem 2 in Ullman [12]. The idea is to polynomially transform the scheduling problem of UET tasks with precedence constraints and no communication delays on m machines to the problem under consideration.

Suppose that we are given an instance $G = (V, E)$ of task graph for this m-processor scheduling problem without communication delay (see the precise definition of this problem in Ullman [12]. Note that this problem can be considered as a special case of problem (PR1) with zero communication delay). The time limit for the m-processor scheduling problem is t, and the task graph G contains exactly $n = mt$ UET tasks. Such a problem is shown to be NP-complete by Ullman [12].

We construct now a task graph $H = (U, A)$ of $19mt$ tasks for 2-processor scheduling problem (with communication delay) with tasks having one or two times units for execution. Our proof differs from that of Ullman [12] only in the construction of task graph H (cf. Figure 2-(a) for a stage of the graph).

- The set of tasks in $H = (U, A)$ is defined by $U = U_1 \cup U_1' \cup U_2$:
  1. $U_1 = \{a_{i,j}, b_{i,j}, c_{i,j}, d_{i,j}, w_{i,j}, x_{i,j}, y_{i,j}, z_{i,j}, r_{i,j}, s_{i,j},\ \ 1 \leq i \leq m,\ 1 \leq j \leq t\}$,
  2. $U_1' = \{a'_{i,j}, b'_{i,j}, c'_{i,j}, d'_{i,j}, x'_{i,j}, y'_{i,j}, z'_{i,j},\ \ 1 \leq i \leq m,\ 1 \leq j \leq t\}$,

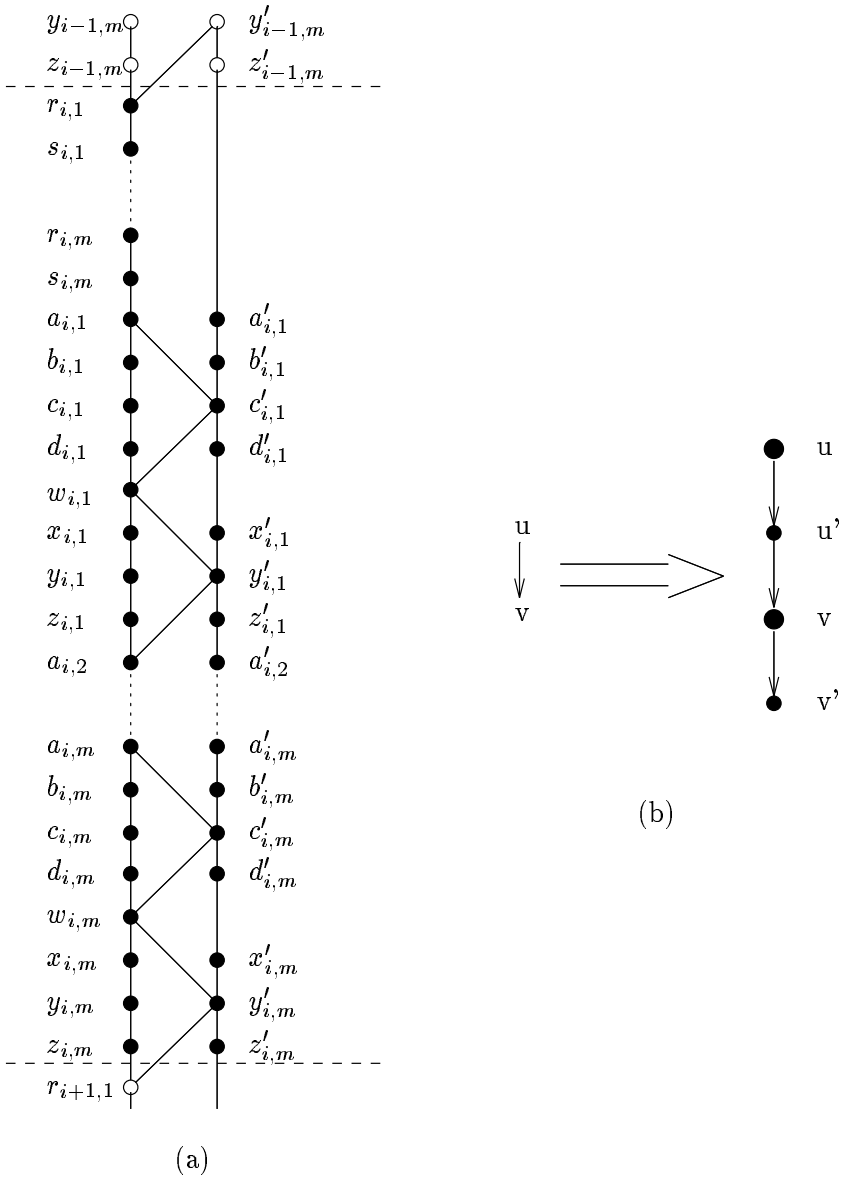


Figure 2: (a): Task graph instance for two processors; (b): the transformation for two tasks $u \rightarrow v$ in precedence relation.

3. $U_2 = \{u, u',\ u \in V\}$.

- The execution time for all tasks is one, except for tasks u which have two time units.

- The precedence relations in the graph H are:

  1. $r_{i,1} \rightarrow s_{i,1} \rightarrow r_{i,2} \rightarrow s_{i,2} \rightarrow \cdots \rightarrow r_{i,m} \rightarrow s_{i,m} \rightarrow a_{i,1}$, for $1 \leq i \leq t$;
  2. $a_{i,1} \rightarrow b_{i,1} \rightarrow c_{i,1} \rightarrow d_{i,1} \rightarrow w_{i,1} \rightarrow x_{i,1} \rightarrow y_{i,1} \rightarrow z_{i,1} \rightarrow a_{i,2} \rightarrow \cdots \rightarrow z_{i,m-1} \rightarrow a_{i,m} \rightarrow b_{i,m} \rightarrow c_{i,m} \rightarrow d_{i,m} \rightarrow w_{i,m} \rightarrow x_{i,m} \rightarrow y_{i,m} \rightarrow z_{i,m}$, for $1 \leq i \leq t$;
  3. $z_{i,m} \rightarrow r_{i+1,1}$, $1 \leq i \leq t-1$;
  4. $a'_{i,1} \rightarrow b'_{i,1} \rightarrow c'_{i,1} \rightarrow d'_{i,1} \rightarrow x'_{i,1} \rightarrow y'_{i,1} \rightarrow z'_{i,1} \rightarrow a'_{i,2} \rightarrow \cdots \rightarrow z'_{i,m-1} \rightarrow a'_{i,m} \rightarrow b'_{i,m} \rightarrow c'_{i,m} \rightarrow d'_{i,m} \rightarrow x'_{i,m} \rightarrow y'_{i,m} \rightarrow z'_{i,m}$, for $1 \leq i \leq t$;
  5. $z'_{i,m} \rightarrow a'_{i+1,1}$, $1 \leq i \leq t-1$;
  6. $a_{i,1} \rightarrow c'_{i,1} \rightarrow w_{i,1} \rightarrow y'_{i,1} \rightarrow a_{i,2} \rightarrow c'_{i,2} \rightarrow w_{i,2} \rightarrow y'_{i,2} \rightarrow \cdots \rightarrow a_{i,m} \rightarrow c'_{i,m} \rightarrow w_{i,m} \rightarrow y'_{i,m}$, for $1 \leq i \leq t$;
  7. $y'_{i,m} \rightarrow r_{i+1,1}$, for $1 \leq i \leq t-1$;
  8. $u \rightarrow u'$ for all $u \in G$,
  9. $u' \rightarrow v$ in H if and only if $u \rightarrow v$ in G.

The time limit for the schedule of H is $10mt$.

Observe that the total execution time of the tasks of H is $20mt$. Thus, in any feasible schedule of H there should be no processor idling. There is a total order on tasks of U_1. Moreover, the total execution time of these tasks is $10mt$. Therefore, these tasks should be assigned to the same processor, say processor 1, in order for a schedule to be feasible. The execution of any of these tasks is critical in the sense that it should be started immediately after the completion of its predecessor in the path.

Tasks of U'_1 and U_2 can only be assigned to processor 2. Due to communications between tasks of U_1 and those of U'_1, the execution of tasks $y'_{i,j} \rightarrow z'_{i,j} \rightarrow a'_{i,j+1} \rightarrow b'_{i,j+1} \rightarrow c'_{i,j+1}$, $1 \leq i \leq t$, $1 \leq j \leq m-1$, is critical too. Therefore, on processor 2,

there are in total mt unit-time slots (inbetween $c'_{i,j}$ and $y'_{i,j}$, $1 \leq i \leq t$, $1 \leq j \leq m$) available for the executions of UET tasks of U_2 (i.e. tasks $u' \in U_2$). The two-unit-time tasks of U_2 (i.e. tasks $u \in U_2$) can only be executed inbetween $y'_{i-1,m}$ and $c'_{i,1}$, which, for each fixed i, $1 \leq i \leq t$, has $2m$ time units in total.

It is now easy to see (due to the precedence relation between u and u') that there is a feasible schedule for H if and only if there is a feasible schedule for G. The detailed arguments are given in Ullman [12]. ∎

Note that the constraint of a single bus is irrelevant, since any feasible schedule contains at most 1 communication in each slot, and thus, there are no bus contentions. Therefore, the above theorem also implies that m-machine scheduling with UCT and no bus constraint is NP-complete when tasks have one or two time unit execution times. Note also that the difference in communication semantics is also irrelevant in the case of two processors. Thus, the above theorem holds for both independent-data and common-data communication semantics.

3.2 Scheduling with Task Preallocation

Consider now the case where task assignments are defined before hand. We will consider the decision problem (PR2) with further restriction that tasks assigned to the same processor are totally ordered. In this case, the scheduling decisions concern the communications on the bus alone.

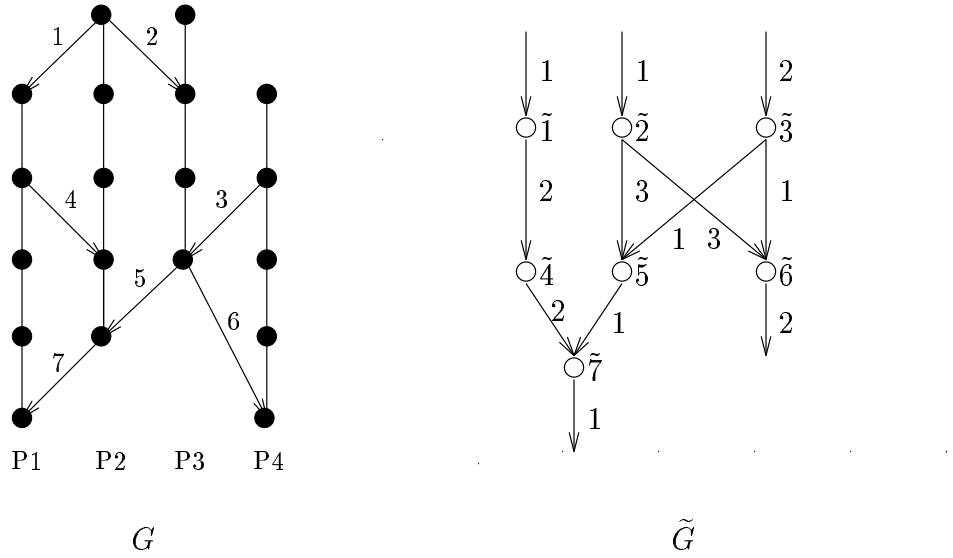
Thus, we will transform this scheduling problem to a dual problem with a single machine (representing the bus) to execute tasks (representing communications). The execution of these (communication) tasks is constrained by precedence delays. More precisely, the single-machine scheduling problem is defined as follows. The task graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ is a weighted graph with vertices weighted by task processing times p_i, $i \in \widetilde{V}$, and arcs weighted by lengths of delays l_{ij}. For any pair of tasks $i, j \in \widetilde{V}$, if $(i,j) \in \widetilde{E}$, then task j can start execution only l_{ij} time units after the execution completion of task i, i.e., $c_i + l_{ij} \leq a_j$, where c_i is the completion time of task i, a_j is the starting time of task j. An initial task $v \in \widetilde{V}$ has release time r_v, i.e., it becomes available r_v time units after the starting time (0). A final task $v \in \widetilde{V}$ has delivery time d_v, i.e. it leaves the system d_v time units after its completion time.

By convention $r_v = 0$ (resp. $d_v = 0$) if v is not an initial (resp. a final) task in $\widetilde{G}$. The purpose of the scheduling is to minimize the makespan of $\widetilde{G}$, defined by $\max_{v \in \widetilde{V}} c_v + d_v$.

Precedence delays in the single-machine scheduling problem are due to task executions on parallel processors in the original scheduling problem. Indeed, if in the parallel-processor scheduling problem, two tasks u and v are to be run on the same processor with l tasks inbetween (including) tasks u and v, where u is run before v, then communications represented by outgoing arcs from v can be carried out only l time units after all communications represented by incoming arcs to u have completed. These l time units correspond to the time interval for running tasks inbetween (including) tasks u and v. Task release (resp. delivery) times in the single-machine scheduling problem correspond to the lengths of chains beginning with initial (resp. ending with final) tasks that send (resp. receive) messages in the parallel-processor scheduling problem. An example is provided in Figure 3 to illustrate the transformation from the parallel-processor scheduling problem G (where tasks of the same column are assinged to the same processor) to the single-machine scheduling problem $\widetilde{G}$ (where the labels of arcs represent the precedence delays and the labels of vertices correspond to the labels of arcs of G).

For the single-machine scheduling problem, we will consider a special class of graphs $\widetilde{\mathcal{G}}$ with UET tasks and integer precedence delays: For any graph $\widetilde{G} = (\widetilde{V}, \widetilde{E}) \in \widetilde{\mathcal{G}}$, let $\widetilde{P}(\cdot)$ and $\widetilde{S}(\cdot)$ denote the sets of immediate predecessors and successors in $\widetilde{G}$.

- There are four types of vertices:
  - $u \in \widetilde{V}$ is of type 0 if $|\widetilde{S}(u)| = 0$;
  - $u \in \widetilde{V}$ is of type 1 if $|\widetilde{S}(u)| = 1$;
  - $u \in \widetilde{V}$ is of type 2 if $|\widetilde{S}(u)| = 2$ and $l_{u,v} \neq l_{u,w}$ for $v, w \in \widetilde{S}(u)$;
  - $u \in \widetilde{V}$ is of type 3 if $|\widetilde{S}(u)| \geq 2$ and for any $v, w \in \widetilde{S}(u)$, $l_{u,v} = l_{u,w}$.
- For any two vertices $u, v \in \widetilde{V}$, either $\widetilde{P}(u) \cap \widetilde{P}(v) = \emptyset$ or $\widetilde{S}(u) \cap \widetilde{S}(v) = \emptyset$.
- For any $(u, v) \in \widetilde{E}$, either $|\widetilde{P}(v)| = 1$ or $|\widetilde{S}(u)| = 1$, and if u is of type 3 then v is either of type 1 or 3.



G $\tilde{G}$

Figure 3: Transformation from parallel-processor scheduling problem G to single-machine scheduling problem $\tilde{G}$.

We define the following single-machine decision problem:

(P1): Single-machine scheduling with unit execution time and integer lengths of precedence delays.

Given a directed acyclic graph $\tilde{G} = (\tilde{V}, \tilde{E}) \in \tilde{\mathcal{G}}$ with unit execution time and positive integer precedence delays $l_{ij} \in I\!N_+$, nonnegative integer release times $r_i \in I\!N \stackrel{\text{def}}{=} \{0, 1, 2, \cdots\}$ and nonnegative integer delivery times $d_i \in I\!N$ for tasks, and a time limit $T \in I\!N_+$, does there exist a function $\sigma : V \to \{0, 1, 2, \cdots, T-1\}$ such that $\sigma(i) + 1 + l_{ij} \leq \sigma(j)$ and $r_j \leq \sigma(j) \leq T - d_j - 1$ for all $(i, j) \in E$?

Lemma 2 *The single-machine problem (P1) is NP-complete.*

Proof. The proof is analogous to (although slightly more involved than) that in Finta and Liu [3]. A detailed proof is given in Appendix A.

Lemma 3 *Problem (P1) polynomially transforms to (PR2).*

Proof. Given an instance graph $\widetilde{G} = (\widetilde{V}, \widetilde{E}) \in \widetilde{\mathcal{G}}$ of problem (P1), we construct an instance graph $G = (V, E)$ for problem (PR2). As we mentioned previously, vertices in $\widetilde{G}$ correspond to communications in G, and arcs in $\widetilde{G}$ correspond to (chains of) tasks in G. An example is illustrated in Figure 4.

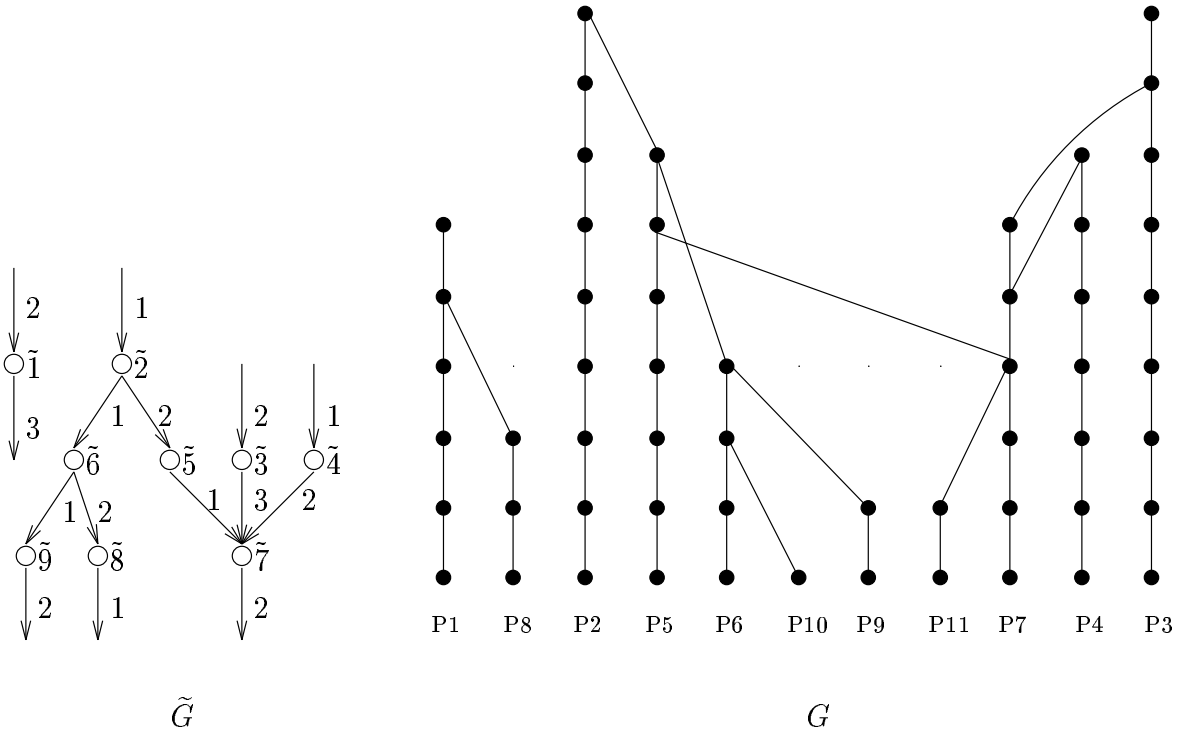


Figure 4: Construction of graph G for parallel-processor scheduling problem (PR2) from an instance $\widetilde{G}$ of single-machine scheduling problem (P1).

For all tasks $u \in \widetilde{V}$, we define $L(u)$, the length of the longest path from vertex u to the final tasks of $\widetilde{G}$:

$$L(u) = \begin{cases} \max_{v \in \widetilde{S}(u)} \left(l_{uv} + 1 + L(v) \right), & \widetilde{S}(u) \neq \emptyset, \\ d_u, & \widetilde{S}(u) = \emptyset. \end{cases}$$

Let $\widetilde{V}_I = \{i_1, i_2, \cdots, i_s\}$ be the set of initial tasks in $\widetilde{G}$. For each task $i_j \in \widetilde{V}_I$, we construct a chain of $r_{i_j} + 1 + L(i_j)$ UET tasks in G, which are assigned to processor

j, $1 \leq j \leq s$. In our example of Figure 4, $\widetilde{V}_I = \{\widetilde{1}, \widetilde{2}, \widetilde{3}, \widetilde{4}\}$, thus, in G we construct 4 chains of UET tasks, chains of lengths $6, 9, 9, 7$ to be run on processors P1, P2, P3 and P4, respectively.

Let $\{\widetilde{E}_1, \widetilde{E}_2, \cdots, \widetilde{E}_e\}$ be a partition of the set of arcs $\widetilde{E}$ such that $(u_1, v_1) \in \widetilde{E}$ and $(u_2, v_2) \in \widetilde{E}$ belong to the same subset $\widetilde{E}_j$, $1 \leq j \leq e$, if and only if $u_1 = u_2$ or $v_1 = v_2$. Note that due to the restrictions made on the class of graphs $\widetilde{\mathcal{G}}$, the relation $u_1 = u_2$ or $v_1 = v_2$ is an equivalence relation defined on the set of arcs $\widetilde{E}$. For each subset $\widetilde{E}_j$, $1 \leq j \leq e$, we construct in G a chain of $\max_{(u,v)\in\widetilde{E}_j} L(u)$ UET tasks which are assigned to processor $s + j$, $1 \leq j \leq e$. For the example in Figure 4, since the partition is $\widetilde{E}_1 = \{(\widetilde{2}, \widetilde{5}), (\widetilde{2}, \widetilde{6})\}$, $\widetilde{E}_2 = \{(\widetilde{6}, \widetilde{8}), (\widetilde{6}, \widetilde{9})\}$ and $\widetilde{E}_3 = \{(\widetilde{3}, \widetilde{7}), (\widetilde{4}, \widetilde{7}), (\widetilde{5}, \widetilde{7})\}$, we construct on processors (P5, P6, P7) three chains of $7, 4, 6$ UET tasks, respectively.

Let $\widetilde{V}_F = \{f_1, f_2, \cdots, f_t\}$ be the set of final tasks of $\widetilde{G}$. For each task $f_j \in \widetilde{V}_F$, we construct a chain of d_{f_j} UET tasks in G, which are assigned to processor $s + e + j$, $1 \leq j \leq t$. In the example of Figure 4, we construct four chains of $3, 2, 1, 2$ UET tasks on the last four processors P8, P9, P10 and P11.

Thus, in the parallel-processor scheduling problem, there are $m = s + e + t$ processors in total, each of which executes a chain of UET tasks of G.

We now define the communications between these chains, which result in communications among the m processors. These communications are defined according to the set of vertices in $\widetilde{V}$. For each $v \in \widetilde{V}$,

Case 1: if $\widetilde{P}(v) = \widetilde{S}(v) = \emptyset$, then there are j and k such that $v \equiv i_j \equiv f_k$. We construct an arc from the r_{i_j}-th task on processor j to the first task on processor $s + e + k$ (e.g. in Figure 4, from the second task on P1 to the first task on P8);

Case 2: if $\widetilde{P}(v) = \emptyset$ and $\widetilde{S}(v) \neq \emptyset$, then there are j and k such that $v \equiv i_j$ and for all $w \in \widetilde{S}(v)$, $(v, w) \in \widetilde{E}_k$. If $|\widetilde{S}(v)| > 1$, then we construct an arc from the r_{i_j}-th task on processor j to the first task on processor $s + k$ (e.g. in Figure 4, from the first task on P2 to the first task on P5). If, however, $\widetilde{S}(v) = \{w\}$, then we construct an arc from the r_{i_j}-th task on processor j to

the $\left(\max_{v' \in \widetilde{P}(w)} l_{v',w} - l_{v,w} + 1\right)$-st task on processor $s + k$ (e.g. in Figure 4, from the first task on P4 to the second task on P7);

Case 3: if $\widetilde{P}(v) \neq \emptyset$ and $\widetilde{S}(v) = \emptyset$, then there are j and k such that $v \equiv f_k$, and for all $u \in \widetilde{P}(v)$, $(u,v) \in \widetilde{E}_j$. We construct an arc from the $\left(\max_{u \in \widetilde{P}(v)} l_{u,v}\right)$-th task on processor $s+j$ to the first task on processor $s+e+k$ (e.g. in Figure 4, from the third task on P7 to the first task on P11);

Case 4: if $\widetilde{P}(v) \neq \emptyset$ and $\widetilde{S}(v) \neq \emptyset$, then then there are j and k such that for all $u \in \widetilde{P}(v)$, $(u,v) \in \widetilde{E}_j$, and for all $w \in \widetilde{S}(v)$, $(v,w) \in \widetilde{E}_k$. If $|\widetilde{S}(v)| > 1$, then we construct an arc from the $\left(\max_{u \in \widetilde{P}(v)} l_{u,v}\right)$-th task on processor $s + j$ to the first task on processor $s + k$ (e.g. in Figure 4, from the first task on P5 to the first task on P6). If, however, $\widetilde{S}(v) = \{w\}$, then we construct an arc from the $\left(\max_{u \in \widetilde{P}(v)} l_{u,v}\right)$-th task on processor $s + j$ to the $\left(\max_{v' \in \widetilde{P}(w)} l_{v',w} - l_{v,w} + 1\right)$-st task on processor $s + k$ (e.g. in Figure 4, from the second task on P5 to the third task on P7).

The construction of task graph G is thus completed. By construction, tasks assigned to the same processor are totally ordered. Without loss of generality we let the tasks to run on the processors as soon as they become available. The corresponding (PR2) decision problem is therefore to answer the question whether there is a schedule for communications on the bus such that the makespan of G is smaller than or equal to T.

It is quite clear that if there is a feasible schedule σ for the single-machine scheduling problem (P1), then the corresponding communications on the bus can be scheduled at the same time epochs, and moreover, the induced schedule τ for the parallel-processor scheduling problem (PR2) has makespan smaller than or equal to T.

We now show that the converse is true. Let there be a feasible schedule τ for the parallel-processor scheduling problem (PR2) on graph G. Let $\tau(u,v)$ denote the starting time of the communication $(u,v) \in E$ on the bus. Let $N = |\widetilde{V}|$ be the number of communications in (PR2). These communication arcs are labeled by

$(u_1, v_1), \cdots, (u_N, v_N) \in E$ such that vertex $j \in \widetilde{V}$ corresponds to communication (u_j, v_j). Assume further that $\tau(u_j, v_j) < \tau(u_{j+1}, v_{j+1})$.

We prove by induction on $n = 1, 2, \cdots, N$ that the schedule σ defined by $\sigma(n) = \tau(u_n, v_n)$ is a feasible schedule of (P1) on $\widetilde{G}$.

For $n = 1$, clearly $\widetilde{P}(1) = \emptyset$ and 1 is an initial task in $\widetilde{G}$, so that there is k such that $i_k = 1$. Thus, by the construction of graph G, $u_1 \in V$ is the r_{i_k}-th task on processor k in problem (PR2). This implies that $\tau(u_1, v_1) = r_{i_k}$. Hence, $\sigma(1) = \tau(u_1, v_1) = r_{i_k}$ is a feasible schedule for task 1 of $\widetilde{G}$.

Assume that for some $n \geq 2$, $\sigma(j) = \tau(u_j, v_j)$ is a feasible schedule of (P1) for tasks $j = 1, 2, \cdots, n-1$ of $\widetilde{G}$. Consider task n of $\widetilde{G}$.

If n is an initial task, then, as in the case $n = 1$, $\sigma(n) = \tau(u_n, v_n)$ is a feasible schedule for task n of $\widetilde{G}$. Assume now that $\widetilde{P}(n) \neq \emptyset$. Let $q \in \widetilde{P}(n)$ be an arbitrary predecessor of n in $\widetilde{G}$. We show that $\sigma(q) + 1 + l_{q,n} \leq \tau(u_n, v_n)$. Clearly, $\widetilde{S}(q) \neq \emptyset$ and $\widetilde{P}(n) \neq \emptyset$. By the construction of graph G, there exists j such that $(q, n) \in \widetilde{E}_j$ (cf. Cases 3 and 4 in the graph construction), and such that v_q and u_n of G are assigned to processor $s + j$. If $|\widetilde{S}(q)| > 1$, then v_q is the first task on processor $s + j$ (cf. Cases 2 and 4 in the graph construction). Since u_n is the $l_{q,n}$-th task on processor $s + j$, we have that

$$\tau(u_n, v_n) \geq \tau(u_q, v_q) + 1 + l_{q,n} = \sigma(q) + 1 + l_{q,n}.$$

If, however, $\widetilde{S}(q) = \{n\}$, then v_q is the $\left(\max_{q' \in \widetilde{P}(n)} l_{q',n} - l_{q,n} + 1\right)$-st task on processor $s + j$ (cf. Cases 2 and 4 in the graph construction). Again, since u_n is the $\left(\max_{q' \in \widetilde{P}(n)} l_{q',n}\right)$-th task on processor $s + j$, we have that

$$\tau(u_n, v_n) \geq \tau(u_q, v_q) + 1 + \max_{q' \in \widetilde{P}(n)} l_{q',n} - \left(\max_{q' \in \widetilde{P}(n)} l_{q',n} - l_{q,n}\right) = \tau(u_q, v_q) + 1 + l_{q,n} = \sigma(q) + 1 + l_{q,n}.$$

Therefore, $\sigma(n) = \tau(u_n, v_n)$ is a feasible schedule for task n of $\widetilde{G}$ in the sense that

$$\sigma(n) = \tau(u_n, v_n) \geq \tau(u_q, v_q) + 1 + l_{q,n} = \sigma(q) + 1 + l_{q,n}.$$

By induction, the feasibility of σ for $\widetilde{G}$ on a single machine is proved.

It is simple to check that schedule σ for $\widetilde{G}$ and schedule τ for G have the same makespan. The proof is therefore completed. ∎

As a consequence of Lemma 3, we obtain

Theorem 3 *Problem (PR2) is NP-complete.*

Corollary 2 *In a multiprocessor system with a single bus, if the task assignments are defined a priori, then the makespan minimization of a task graph with UET-UCT and independent-data communication semantics is NP-hard.*

4 Scheduling with Common-Data Communication Semantics

In this section we consider scheduling problems with common-data communication semantics. The results are similar to those of Section 3.

Consider first the case where tasks have no preallocations. As in Section 3.1, we prove that the problem is NP-hard even for four processors with UET, and for two processors with two different units of task execution times.

Lemma 4 *The partition problem polynomially transforms to (PR3) with UET and four processors.*

Proof. The arguments of the proof are similar to that of Lemma 1. We provide here only a sketch of the proof.

Given an instance of the partition problem (see the paragraph before Lemma 1), we construct the following instance of problem (PR3) with UET and four processors, such that there exists a feasible schedule σ if and only if the partition problem has a solution.

Let $A = \{a_1, \cdots, a_n\}$ be the set of items in the partition problem. The time limit for the scheduling function is $T = 2B + 4$. The task graph $G = (V, E)$ is constructed as follows.

- There is a chain of T UET tasks, denoted by $x_1 \rightarrow x_2 \rightarrow \cdots \rightarrow x_T$, and $T - 5$ tasks with UET, denoted by $y_6, y_7, \cdots, y_T$.

- For $4 \leq i \leq T-2$, there is an arc from task x_i to task y_{i+2}.
- For each a_i, $1 \leq i \leq n$, there is a chain of $2s(a_i)$ tasks with UET, denoted by $z_{i,1} \rightarrow z_{i,2} \rightarrow \cdots \rightarrow z_{i,2s(a_i)}$, tasks z_{i1} being immediate successors of task x_2 (second task in the x-chain).

The total number of tasks in the graph is $|V| = 4T - 13$. Figure 5 illustrates the task graph that we construct for each instance of partition problem.

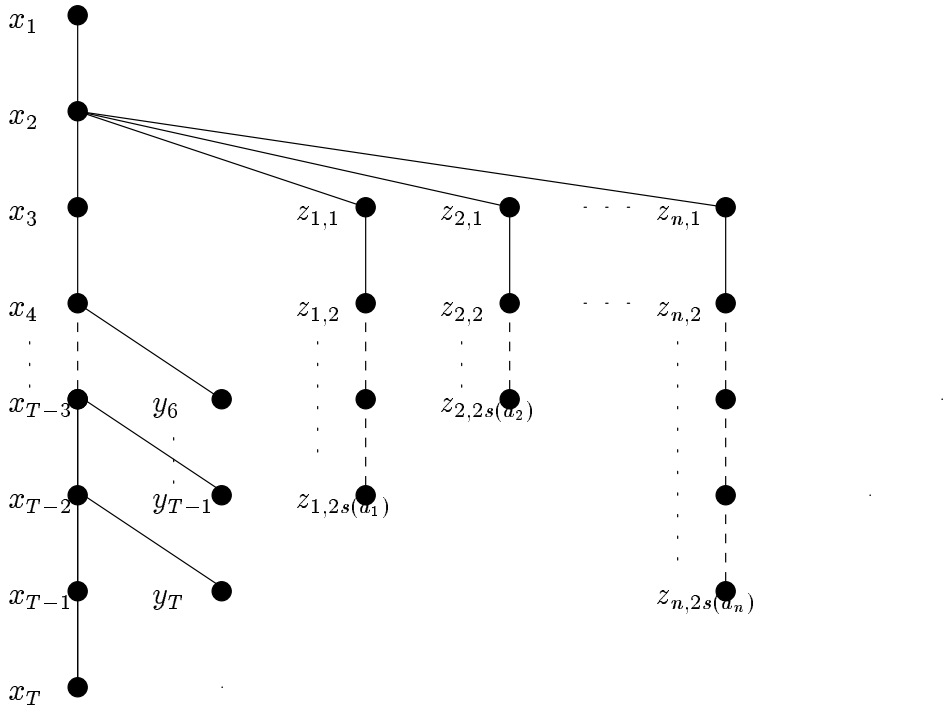


Figure 5: Task graph instance of (PR3) corresponding to the instance of partition problem.

As argued in the proof of Lemma 1, all x-tasks are assinged to processor 1, and no other task successor of x-tasks can be assigned to processor 1. Thus, there are $T-5$ communications corresponding to the arcs $x_i \rightarrow y_{i+2}$, $4 \leq i \leq T-2$, to be done on the bus in the time interval $[4, T-1)$. These communications are critical in the sense that the bus has to send the message $x_i \rightarrow y_{i+2}$ as soon as task x_i completes

in any feasible schedule. Hence, communications $x_2 \rightarrow z_{i1}$ (when processor 1 sends data), $i = 1, 2, \ldots, n$, can be dealt with by the bus only in the time interval $[2, 4)$. Therefore, z-tasks can be assigned only to at most two processors different from processor 1 (say processors 3 and 4). Moreover, there should be no interprocessor communications between processors 3 and 4 which execute tasks of z-chains, so that task assignment should be such that the z-chains are not broken.

Due to common-data communication semantics, there are only two inter-processor communications for arcs $x_2 \rightarrow z_{i1}$: one between processors 1 and 3, another between processors 1 and 4. Thus, a feasible schedule exists if and only if there is a subset $A' \subset A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a) = B$. ∎

Theorem 4 *The decision problem (PR3) with 4 processors is NP-complete.*

Corollary 3 *In a multiprocessor system with a single bus, the makespan minimization of a task graph with UET-UCT and common-data communication semantics is NP-hard, even if there are only 4 processors.*

As remarked in the previous section, the difference in communication semantics is irrelevant in the case of two processors, so that Theorem 2 still holds.

Consider now the case that tasks are assigned to processors a priori. The result of NP-completeness in Section 3.2 is still valid. Indeed, in the proof, the task graph and task assignment are such that no processor executes two tasks which are immediate successors of the same task. Thus, the common-data communication semantics makes no difference in the proof.

Theorem 5 *Problem (PR4) is NP-complete.*

Corollary 4 *In a multiprocessor system with a single bus, if the task assignments are defined a priori, then the makespan minimization of a task graph with UET-UCT and common-data communication semantics is NP-hard.*

5 Extensions

In this section, we discuss how our results can be extended to other communication mechanisms. In particular, we consider blocking communications and broadcasting communications.

5.1 Communications with Blocking

In the previous sections, we discussed the mechanism of communications without blocking. We now consider *communications with blocking*. Communication with blocking means that when a processor is sending (resp. receiving) a message, it stops processing a task until the communication completes (resp. when the communication starts until it completes). The NP-completeness results obtained in Sections 3 and 4 remain true. We discuss below how these can be shown for problem (PR2) in the case that blocking occurs only for message sending. The other cases can be dealt with in a similar way.

In order to prove that problem (PR2) with blocking in message sending is NP-complete, we need

Lemma 5 *Problem (P1) polynomially transforms to (PR2) with blocking in message sending.*

Proof. The proof is similar to that of Lemma 3. In Appendix B, we construct an instance graph $G = (V, E)$ for problem (PR2) from an given instance graph $\widetilde{G} = (\widetilde{V}, \widetilde{E}) \in \widetilde{\mathcal{G}}$ of problem (P1). ∎

Theorem 6 *Problem (PR2) with blocking in message sending is NP-complete.*

Note that communication with blocking in message sending can also be such that the sender stops processing tasks until the communication starts. The same results hold.

5.2 Broadcasting Communications

In the case of common-data communication semantics, broadcasting communication mechanism can be used so that a message can have multiple destinations and that a message can have several receivers simultaneously. In this case, problems (PR3) and (PR4) are still NP-complete.

Theorem 7 *The decision problem (PR3) with 4 processors and broadcasting communications is NP-complete.*

Theorem 8 *Problem (PR4) with broadcasting communications is NP-complete.*

In order to prove Theorem 7, it suffices to note that the proof of Lemma 1 works in changing the construction of the task graph so that there are arcs $x_3 \rightarrow z_{i1}$ instead of $x_2 \rightarrow z_{i1}$.

To show Theorem 8, we need

Lemma 6 *Problem (P1) polynomially transforms to (PR4) with broadcasting communications.*

Proof. The proof is similar to that of Lemma 3. In Appendix C, we construct an instance graph $G = (V, E)$ for problem (PR4) from an given instance graph $\widetilde{G} = (\widetilde{V}, \widetilde{E}) \in \widetilde{\mathcal{G}}$ of problem (P1). ∎

6 Concluding Remarks

In this paper, we have analyzed scheduling problems in a single-bus multiprocessor system. The goal of these scheduling problems is to minimize the makespan when parallel programs are represented by task graphs. We have shown that these problems are NP-hard even under very specific assumptions, e.g. UET and UCT. Thus, the general scheduling problem of parallel programs with communication resource constraints is NP-hard.

We have considered several variants of the problem: tasks with or without preallocation, communications with independent-data semantics or common-data semantics. Our results have been extended to the cases of blocking communications and

broadcasting communications. Thus, the NP-hardness of analyzed problems extends of course to scheduling problems with general communication semantics and communication mechanisms.

It is easily seen that our results applies also to multiprocessor systems with shared memory.

Appendices

A Proof of Lemma 3

As discussed in [3], the release times and delivery times have no effect on the complexity of problem (P1). Indeed, it suffices to add a fictitious initial task and a fictitious final task in the task graph such that all tasks are successors of the fictitious initial task and predecessors of the fictitious final task. The corresponding precedence delays are the release times (for arcs beginning with the fictitious initial task) and delivery times (for arcs ending with the fictitious final task). The task graph thus obtained will therefore have no release times and delivery times. Thus, in what follows, we consider problem (P1) where release times and delivery times are all zero.

In order to show that problem (P1) is NP-complete, we begin with introducing a slightly more complex problem (P2) which can be polynomially transformed to (P1). In (P2) there are some forbidden regions for the scheduling function, i.e., the machine is not available in some periods of time. We then show this new problem (P2) to be NP-complete so that (P1) is also NP-complete.

> *(P2): Single-machine scheduling of graphs of class $\widetilde{\mathcal{G}}$ with unit execution time, integer lengths of precedence delays and forbidden regions.*
>
> Given a directed acyclic graph $\widetilde{G} = (\widetilde{V}, \widetilde{E}) \in \widetilde{\mathcal{G}}$ with unit execution time $p_j = 1$ for all $j \in \widetilde{V}$, precedence delays $l_{ij} \in I\!N_+$, a time limit $T \in I\!N_+$, and some positive integers $0 \leq b_1 < e_1 < b_2 < e_2 < \cdots < b_r < e_r < T$, does there exist a function $\sigma : \widetilde{V} \rightarrow \{0, 1, \cdots, T-1\}$ such that

(i) for all $(i,j) \in \widetilde{E}$, $\sigma(i) + 1 + l_{ij} \leq \sigma(j)$, and

(ii) for all $i \in \widetilde{V}$ and all $s \in \{1, 2, \cdots, r\}$, $\sigma(i) \notin [b_s, e_s)$?

Lemma 7 *Problem (P2) polynomially transforms to problem (P1).*

Proof. Let

$$V^\circ = \bigcup_{1 \leq s \leq r} \{b_s, b_s + 1, \cdots, e_s - 2, e_s - 1\}.$$

Let $V' = \{i_1, i_2, \cdots, i_h\}$ (resp. $V'' = \{j_1, j_2, \cdots, j_k\}$) be the odd (resp. even) numbers in V° such that $V^\circ = V' \cup V''$ and $i_1 < i_2 < \cdots < i_h$, $j_1 < j_2 < \cdots < j_k$. We construct two chains $\widetilde{G}_1 = (\widetilde{V}_1, \widetilde{E}_1)$ and $\widetilde{G}_2 = (\widetilde{V}_2, \widetilde{E}_2)$ such that

$$\begin{aligned}
\widetilde{V}_1 &= \{v_{i_1}, v_{i_2}, \cdots, v_{i_h}\}, \\
\widetilde{E}_1 &= \{(v_{i_s}, v_{i_{s+1}}) \mid s = 1, 2, \cdots, h-1\}, \\
\widetilde{V}_2 &= \{v_{j_1}, v_{j_2}, \cdots, v_{j_k}\}, \\
\widetilde{E}_2 &= \{(v_{j_s}, v_{j_{s+1}}) \mid s = 1, 2, \cdots, k-1\}.
\end{aligned}$$

The lengths of precedence delays are defined as follows:

$$\begin{aligned}
l_{v_{i_s}, v_{i_{s+1}}} &= i_{s+1} - i_s - 1, \qquad s = 1, 2, \cdots, h-1, \\
l_{v_{j_s}, v_{j_{s+1}}} &= j_{s+1} - j_s - 1, \qquad s = 1, 2, \cdots, k-1.
\end{aligned}$$

Now, for any given instance of (P2) with task graph $\widetilde{G}$ and time limit T, we construct an instance of (P1) as follows. Task graph $\widetilde{G}'$ in (P1) is defined as the union of $\widetilde{G}$, $\widetilde{G}_1$ and $\widetilde{G}_2$ connected by an initial task a_1 and a final task a_2 in such a way that

- the precedence delays between a_1 and any tasks of $\widetilde{G}$ without predecessors are 1;
- the precedence delays between any tasks of $\widetilde{G}$ without successors and a_2 are 1;
- the precedence delay between a_1 and v_{i_1} (resp. v_{j_1}) is $i_1 + 1$ (resp. $j_1 + 1$);

- the precedence delay between v_{i_h} (resp. v_{j_k}) and a_2 is $T-i_h+1$ (resp. $T-j_k+1$).

The time limit in (P1) is T+4. Note that task graph $\widetilde{G}'$ is of class $\widetilde{\mathcal{G}}$.

It is easy to see that there is a solution to (P2) if and only if there is a solution to (P1). Indeed, according to the construction of $\widetilde{G}'$, the chains $\widetilde{G}_1$ and $\widetilde{G}_2$ are critical paths in $\widetilde{G}'$ so that task v_{i_s}, $1 \leq s \leq h$, (resp. v_{j_s}, $1 \leq s \leq k$) should be executed at time $i_s + 2$ (resp. $j_s + 2$). ∎

We now transform the classical 3-satisfiability problem, denoted by 3SAT, to (P2) by a polynomial transformation. Recall the definition of 3SAT:

3SAT: 3-*satisfiability.* Given a set X of binary variables x_i, $1 \leq i \leq m$, and a collection C of clauses c_j over X, $1 \leq j \leq k, |c_j| = 3$, is there a satisfying truth assignment for C?

Lemma 8 *3SAT polynomially transforms to (P2).*

Proof. Given an instance of 3SAT as above, we construct the following instance of (P2), such that there exists a scheduling function σ if and only if 3SAT has a solution.

The structure of our task graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ is similar to the one used by Ullman [12] in the proof of NP-hardness of makespan minimization of UET tasks on identical machines under precedence constraints. In words, task graph $\widetilde{G}$ is constructed as follows: For each variable $x_i \in X$, $1 \leq i \leq m$, we have two paths $x_{i,0} \rightarrow x_{i,1} \rightarrow \cdots \rightarrow x_{i,m}$ and $\bar{x}_{i,0} \rightarrow \bar{x}_{i,1} \rightarrow \cdots \rightarrow \bar{x}_{i,m}$ in $\widetilde{G}$. The arcs $(x_{i,j-1}, x_{i,j})$ and $(\bar{x}_{i,j-1}, \bar{x}_{i,j})$ have precedence delays $l_{x_{i,j-1},x_{ij}} = l_{\bar{x}_{i,j-1},\bar{x}_{ij}} = 2m+j$ for $1 \leq i \leq m$, $1 \leq j \leq m$. For each path there is one more vertex y_i or $\bar{y}_i$, without outgoing arc, connected to the path by an arc $(x_{i,i-1}, y_i)$ or $(\bar{x}_{i,i-1}, \bar{y}_i)$ with precedence delays $l_{x_{i,i-1},y_i} = l_{\bar{x}_{i,i-1},\bar{y}_i} = m$ for $1 \leq i \leq m$. For each clause $c_r \in C$, $1 \leq r \leq k$, we have in $\widetilde{G}$ seven vertices c_{rs}, $1 \leq s \leq 7$. There is an arc from $x_{i,m}$ (or $\bar{x}_{i,m}$) to d^p_{rs} and from d^p_{rs} each clause $c_{r,s}$ whenever it contributes, $1 \leq p \leq 3$.

An example of the construction of the instance of (P2) is illustrated in Figure 6, where the set of literals is $X = \{x_1, x_2, x_3, x_4\}$ and the clauses are $C_1 = x_1 + x_2 + \bar{x}_3$ and $C_2 = \bar{x}_1 + x_3 + \bar{x}_4$, hence, $k = 2$, $m = 4$. The graph is top-bottom oriented and

all the arcs at the same level have the same length. Some vertices and vertex names, and some arcs connected to clause vertices $c_{2,j}$ are omitted for sake of simplicity, $1 \leq j \leq 7$.

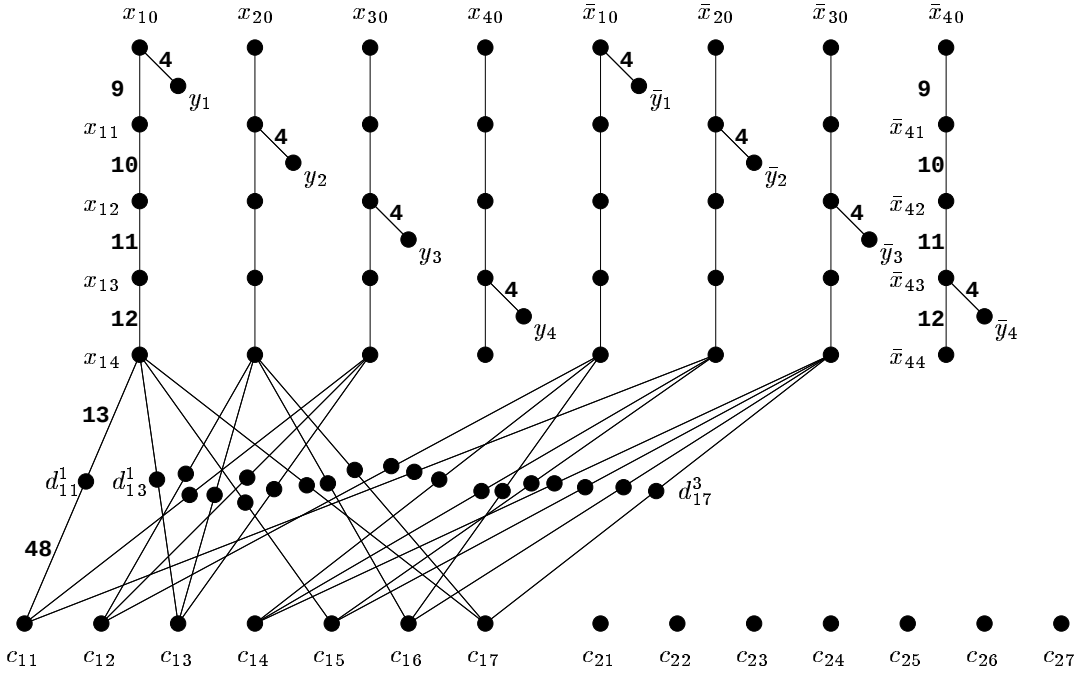


Figure 6: Task graph of the instance (P2) corresponding to the instance of 3SAT.

The formal definition of the graph is the following.

- The set of vertices $\widetilde{V}$ contains:
  - x_{ij} and $\bar{x}_{ij}$ for $1 \leq i \leq m$, $0 \leq j \leq m$,
  - y_i and $\bar{y}_i$ for $1 \leq i \leq m$,
  - d^p_{rs} for $1 \leq r \leq k$, $1 \leq s \leq 7$, $1 \leq p \leq 3$,
  - c_{rs} for $1 \leq r \leq k$ and $1 \leq s \leq 7$.
- The set of arcs $\widetilde{E}$ and precedence delays are:

- $(x_{i,j-1}, x_{ij})$ and $(\bar{x}_{i,j-1}, \bar{x}_{ij})$ with precedence delays $l_{x_{i,j-1},x_{ij}} = l_{\bar{x}_{i,j-1},\bar{x}_{ij}} = 2m + j$ for $1 \leq i \leq m$, $1 \leq j \leq m$,
- $(x_{i,i-1}, y_i)$ and $(\bar{x}_{i,i-1}, \bar{y}_i)$ with precedence delays $l_{x_{i,i-1},y_i} = l_{\bar{x}_{i,i-1},\bar{y}_i} = m$ for $1 \leq i \leq m$.
- The arcs connecting x_{im} (resp. $\bar{x}_{im}$) to d^p_{rs} and d^p_{rs} to c_{rs}, $1 \leq i \leq m$, $1 \leq r \leq k$, $1 \leq s \leq 7$, $1 \leq p \leq 3$, are defined as follows. Let c_r consist of literals $z_{u_1}, z_{u_2}, z_{u_3}$, where each z independently stands for x or $\bar{x}$, in a fixed order, i.e. $c_r = z_{u_1} + z_{u_2} + z_{u_3}$, $1 \leq u_1 < u_2 < u_3 \leq m$, $1 \leq r \leq k$. Let $a_1a_2a_3$ be the binary representation of s, $1 \leq s \leq 7$. Then for $1 \leq p \leq 3$, if $a_p = 1$, we have an arc $(z_{u_p m}, d^p_{rs})$ and (d^p_{rs}, c_{rs}), else, if $a_p = 0$, then we have an arc $(\bar{z}_{u_p m}, d^p_{rs})$ and (d^p_{rs}, c_{rs}), where $\bar{z}$ stands for $\bar{x}$ or x, should z be x or $\bar{x}$, respectively. Note that since a clause should have at least one literal having true assignment, the case $a_1 = a_2 = a_3 = 0$ cannot occur.
  The precedence delays are defined by $l_{z_{u_p m},d_{rs}} = l_{\bar{z}_{u_p m},d_{rs}} = 2m + 3k - 1$, and by $l_{d^p_{rs},c_{rs}} = m + 22k$.

There are $m+4$ forbidden regions for the scheduling function σ. For $1 \leq i \leq m-1$, the i-th forbidden region $F_i = [b_i, e_i)$ is of length i. The last five forbidden regions $F_i = [b_i, e_i)$, $m \leq i \leq m+4$, are of lengths $3k$, $m-1$, $m+k$, $3k$, $m+18k$ respectively. The start and the end of those regions are:

$$b_i = 2m + 1 + (2m + 2)i + \tfrac{i(i-1)}{2}, \qquad e_i = 2m + 1 + (2m + 2)i + \tfrac{i(i+1)}{2}, \quad 1 \leq i \leq m - 1,$$

$$\begin{aligned}
b_m &= \tfrac{5m^2+5m}{2}, & e_m &= \tfrac{5m^2+5m}{2} + 3k,\\
b_{m+1} &= \tfrac{5m^2+7m}{2} + 3k, & e_{m+1} &= \tfrac{5m^2+9m}{2} + 3k - 1,\\
b_{m+2} &= \tfrac{5m^2+9m}{2} + 6k - 1, & e_{m+2} &= \tfrac{5m^2+11m}{2} + 7k - 1,\\
b_{m+3} &= \tfrac{5m^2+11m}{2} + 25k - 1, & e_{m+3} &= \tfrac{5m^2+11m}{2} + 28k - 1,\\
b_{m+4} &= \tfrac{5m^2+11m}{2} + 29k - 1, & e_{m+4} &= \tfrac{5m^2+13m}{2} + 47k - 1.
\end{aligned}$$

Thus, there are $m + 5$ active regions for the machine, the first one with length $4m + 3$, the next $m - 2$ ones with length $2m + 2$, the m-th with length $m + 1$, the $m + 1$-st with length m, the $m + 2$-nd with length $3k$, the $m + 3$-rd with length $18k$, the $m + 4$-th with length k and the $m + 5$-th with length $6k$.

The time limit is $T = \frac{5m^2+13m}{2} + 53k - 1$. Note that $T = |\widetilde{V}| + \sum_{i=0}^{m+4}(e_i - b_i)$ so that under any feasible scheduling solution the machine never idles.

The forbidden and active regions of the machine corresponding to the example of Figure 6 is illustrated in Figure 7. The time limit is $T = 171$.

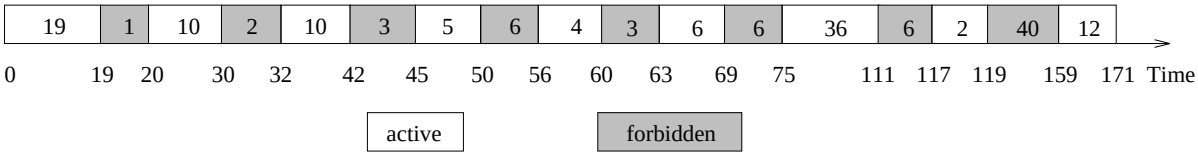


Figure 7: Forbidden and active regions of the machine of the instance (P2).

We claim that there is a solution to the instance of 3SAT if and only if there there is a feasible schedule for the above instance of (P2). The intuitive idea behind the proof is that x_i (or $\bar{x}_i$) is true if and only if the execution of x_{i0} (or $\bar{x}_{i0}$) begins in the time interval $[0, m-1]$. The problem instance of (P2) is constructed in such a way that there is a solution to the instance of 3SAT if and only if there there is a feasible *nonidle* schedule for the above instance of (P2). In order to have a nonidle solution, we have to schedule in the first m time slots either task x_{i0} or $\bar{x}_{i0}$, corresponding to the true value associated to each literal x_i being 1 or 0, respectively. The delays on the precedence constraints are chosen such that once all $x_{i,0}$'s and $\bar{x}_{i,0}$'s are executed, we cannot change the order of execution for their successors, i.e. $x_{i,j}$'s and $\bar{x}_{i,j}$'s, for any fixed j, $1 \leq j \leq m$, without introducing at least one idle time in the schedule.

In the following, we denote by x'_{ij} (resp. y'_i) the first executed task among tasks x_{ij} and $\bar{x}_{ij}$ (resp. y_i and $\bar{y}_i$) and by x''_{ij} (resp. y''_i) the second one. In order to simplify the proof (and the notation in the proof), we consider an additional forbidden region of zero length $F_0 = [b_0, e_0)$ with $b_0 = e_0 = 2m + 1$. We have therefore $m + 5$ forbidden regions in total (including the above one with zero length): $F_0, F_1, \cdots, F_{m+4}$. Similarly, we consider $m+6$ actives regions $A_0, A_1, \cdots, A_{m+5}$, where A_0 is the active region between time zero and time $2m + 1$, and A_1 is the active region from time $2m + 1$ to time $4m + 3$.

For all tasks $v \in \widetilde{V}$, we define $L(v)$, the length of the longest path to vertex v, as follows:

$$L(v) = \begin{cases} \max_{u \in \widetilde{P}(v)} L(u) + l_{uv} + 1, & \widetilde{P}(v) \neq \emptyset, \\ 0, & \widetilde{P}(v) = \emptyset, \end{cases}$$

where $\widetilde{P}(v)$ denotes the set of immediate predecessors of v in the task graph.

A task is said to be available at some time t if each of its predecessors has finished execution and the precedence delay from the predecessor to the task has elapsed by time t.

Claim 1: *Tasks with labels x_{ij} or $\bar{x}_{ij}$ which are successors of the first m executed tasks must be executed as soon as they become available in order to have a nonidle schedule.*

Proof of Claim 1:

Tasks x_{im} or $\bar{x}_{im}$ are not available before time $e_{m-1}+1$ due to the fact that

$$L(x_{im}) = L(\bar{x}_{im}) = \frac{5m^2+3m}{2} = e_{m-1}+1.$$

Since the number of time units where the machine is active until time $e_{m-1}+1$ is $2m^2+2m$, which is equal to the maximum number of possibly available tasks until this time $\{x_{ij}, \bar{x}_{ij}, y_{j+1}, \bar{y}_{j+1},\ 1 \le i \le m,\ 0 \le j \le m-1\}$, all these tasks should be executed by time e_{m-1} in order to have a nonidle schedule.

Let $x'_{i_0 0}$ be the task executed at time 0. It is clear that only task $x'_{i_0 m}$ is available at time $e_{m-1}+1$, provided all tasks on the path $x'_{i_0 0}, x'_{i_0 1}, \cdots, x'_{i_0 m}$ are executed as soon as they become available. Let $x_{i_v 0}$ (or $\bar{x}_{i_v 0}$) be the task executed at time v, $1 \le v \le m-1$. The same argument shows that only task $x_{i_v m}$ (or $\bar{x}_{i_v m}$) is available at time $e_{m-1}+1+v$, provided all tasks on the path $x_{i_v 0}, x_{i_v 1}, \cdots, x_{i_v m}$ (or $\bar{x}_{i_v 0}, \bar{x}_{i_v 1}, \cdots, \bar{x}_{i_v m}$) are executed as soon as they become available. □

Claim 2: *In order to have a nonidle schedule, tasks are executed in active region A_0 in the order of*

$$x'_{i_1 0}, x'_{i_2 0}, \cdots, x'_{i_m 0}, x''_{10}, x''_{20}, \cdots, x''_{m0}, y'_1,$$

where $\{i_1, \cdots, i_m\}$ is a permutation on $\{1, 2, \cdots, m\}$, and in active region A_j, $1 \le j \le m-1$, tasks are executed in the order of

$$y''_j, x'_{i_1 j}, x'_{i_2 j}, \cdots, x'_{i_m j}, x''_{1j}, x''_{2j}, \cdots, x''_{mj}, y'_{j+1},$$

and in active region A_m, tasks are executed in the order of

$$y''_m, x'_{i_1 m}, x'_{i_2 m}, \cdots, x'_{i_m m},$$

and in active region A_{m+1}, tasks are executed in the order of

$$x''_{1m}, x''_{2m}, \cdots, x''_{mm}.$$

Proof of Claim 2:

For $1 \leq i \leq m$, $1 \leq j \leq m$, the length of the path to tasks x_{ij} and $\bar{x}_{ij}$ is:

$$L(x_{ij}) = L(\bar{x}_{ij}) = (2m+2)j + \frac{j(j-1)}{2} = e_{j-1} + 1,$$

and the length of the path to tasks y_j and $\bar{y}_j$ is:

$$L(y_j) = L(\bar{y}_j) = (2m+2)(j-1) + \frac{(j-2)(j-1)}{2} + m + 1.$$

Therefore, the maximum number of possibly available tasks by time $e_{j-1}+1$ is $(2m+2)j$, $1 \leq j \leq m$. Since the number of time units where the machine is active until time $e_{j-1}+1$ is also equal to $(2m+2)j$, $1 \leq j \leq m$, all the tasks should be executed by time e_{j-1} in order to have a nonidle schedule.

At time $b_{j-1} - 1 = (2m+2)(j-1) + 2m + \frac{(j-2)(j-1)}{2}$, $1 \leq j \leq m$, tasks x_{ij} or $\bar{x}_{ij}$ are not available, and task y'_j is available if:

$$\sigma(x'_{j0}) + L(y'_j) \leq b_{j-1} - 1,$$

so that

$$\sigma(x'_{j0}) \leq m - 1, \qquad 1 \leq j \leq m.$$

Therefore, tasks $\{x'_{i0}, 1 \leq i \leq m\}$ should be executed in the first m time slots of the schedule in order not to have idle.

Consider now the first time slot of active region A_j, $1 \leq j \leq m$, i.e. time slot $e_{j-1} = b_{j-1} + j - 1$. Task y''_j is available only if

$$\sigma(x''_{j0}) + L(y''_j) \leq e_{j-1},$$

or equivalently,

$$\sigma(x''_{j0}) \leq m + j - 1.$$

An induction on $j = 1, 2, \cdots, m$ yields that task x''_{j0} is scheduled at $m + j - 1$.

Once the schedule of tasks x'_{i0} and x''_{i0}, $1 \leq i \leq m$, are fixed, a simple inductive argument shows that all tasks x_{ij} or $\bar{x}_{ij}$, $j \geq 1$, should be executed as soon as they become available.

It then follows that, in active region A_0, tasks are executed in the order of

$$x'_{i_1 0}, x'_{i_2 0}, \cdots, x'_{i_m 0}, x''_{10}, x''_{20}, \cdots, x''_{m0}, y'_1,$$

where $\{i_1, \cdots, i_m\}$ is a permutation on $\{1, 2, \cdots, m\}$, in active region A_j, $1 \leq j \leq m - 1$, tasks are executed in the order of

$$y''_j, x'_{i_1 j}, x'_{i_2 j}, \cdots, x'_{i_m j}, x''_{1j}, x''_{2j}, \cdots, x''_{mj}, y'_{j+1},$$

and in active region A_m, tasks are executed in the order of

$$y''_m, x'_{i_1 m}, x'_{i_2 m}, \cdots, x'_{i_m m},$$

and in active region A_{m+1}, tasks are executed in the order of

$$x''_{1m}, x''_{2m}, \cdots, x''_{mm}.$$

□

Claim 3 *In order to have a nonidle schedule, only tasks of type d^p_{rs} and c_{rs}, $1 \leq r \leq k$, $1 \leq s \leq 7$, $1 \leq p \leq 3$, are executed in the last four active regions.*

Proof of Claim 3:

The length of the path to tasks of type d^p_{rs} is

$$L(d^p_{rs}) = \frac{5m^2 + 7m}{2} + 3k = b_{m+1}.$$

Hence no task of this type is available before the end of the active region A_{m+1}, nor their successors tasks of type c_{rs}. Since the number of tasks to be executed in the last four active regions is equal to the number of tasks of type d^p_{rs} and c_{rs}, only those type of tasks are to be executed in the last four active regions in order to have a nonidle schedule. □

Claim 4 *In order to have a nonidle schedule, tasks of type* d^p_{rs}, $1 \leq r \leq k$, $1 \leq s \leq 7$, $1 \leq p \leq 3$, *that are available for execution in the active region* A_{m+2} *should have their predecessors executed in the active region* A_m.

Proof of Claim 4:

The earliest time when some successor of the first executed task of type x''_{jm} in the active region A_{m+1} becomes available is:

$$e_m + 1 + l_{x''_{jm}, d^p_{rs}} = \frac{5m^2 + 9m}{2} + 6k = b_{m+2} + 1.$$

Thus, in order for some task d^p_{rs} to be executed in the active region A_{m+2}, its predecessor, i.e. task of type x_{jm} or $\bar{x}_{jm}$, should be executed in the active region A_m. It then follows that the predecessors of tasks that are available for execution in the active region A_{m+2} are of type x'_{jm}. □

Claim 5 *In order to have a nonidle schedule, tasks of type* c_{rs}, $1 \leq r \leq k$, $1 \leq s \leq 7$, *that are available for execution in the active region* A_{m+4} *should have their predecessors executed in the active region* $A_m + 2$.

Proof of Claim 5:

The earliest time when some successor of the first executed task of type d^p_{rs} in the active region A_{m+3} may become available is:

$$e_{m+2} + 1 + l_{d^p_{rs}, c_{rs}} = \frac{5m^2 + 13m}{2} + 29k = b_{m+4} + m + 1.$$

Thus, in order for some task c_{rs} to be executed in the active region A_{m+4}, all its predecessors, i.e. tasks of type d^p_{rs}, should be executed in the active region A_{m+2}. It then follows, using Claim 4 that the predecessors of tasks that are available for execution in the active region A_{m+4} are of type x'_{jm}. □

Observe that for each pair c_{rs} and c_{rt}, $s \neq t$, there is at least one j such that either x_{jm} precedes c_{rs} and $\bar{x}_{jm}$ precedes c_{rt}, or $\bar{x}_{jm}$ precedes c_{rs} and x_{jm} precedes c_{rt}. Thus, for any r, $1 \leq r \leq k$, one and only one of the seven tasks c_{rs}, $1 \leq s \leq 7$, can be executed in the active region A_{m+2}.

Therefore, we have nonidling scheduling function σ if and only if the first m executed tasks correspond to a satisfying truth assignment. ∎

Theorem 9 *Both problems (P1) and (P2) are NP-complete.*

Proof. The assertion follows from Lemmas 7 and 8. ∎

B Proof of Lemma 5

Observe first that in a problem (PR2) with blocking in message sending, the communication graph differs from that of the problem (PR2) without blocking in that message sendings from the same processor have precedence relations.

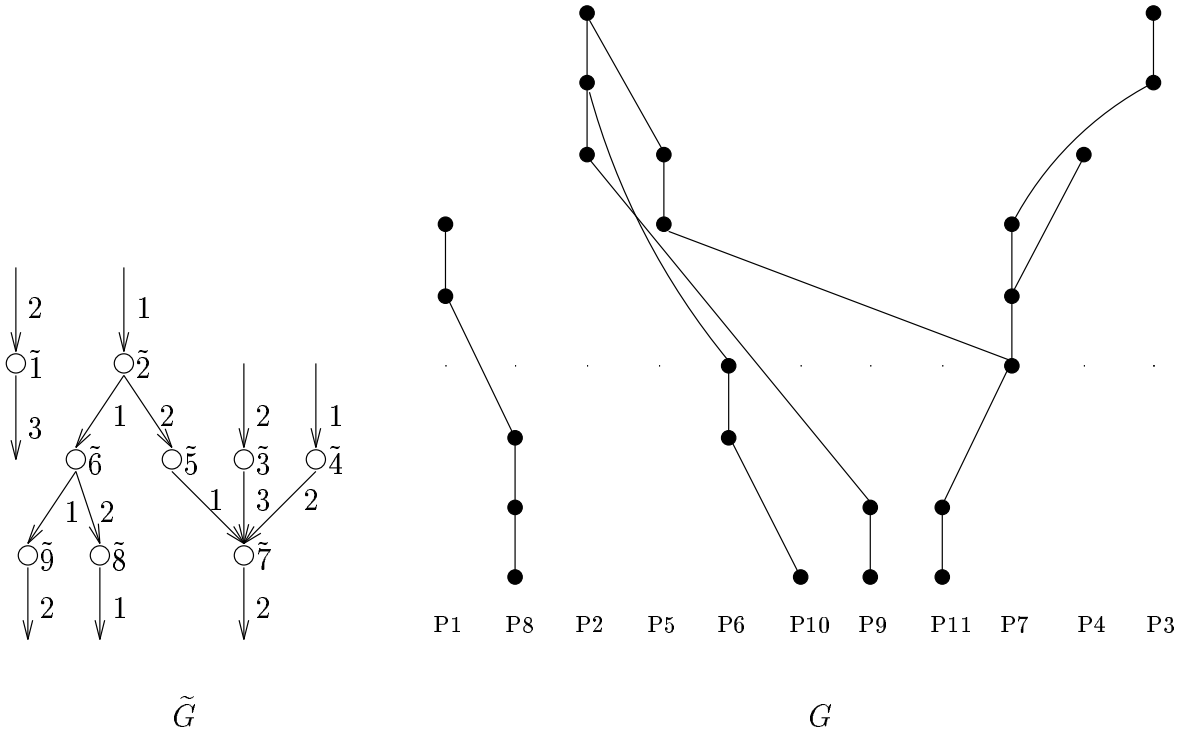
We construct an instance graph $G = (V, E)$ for problem (PR2) from an given instance graph $\widetilde{G} = (\widetilde{V}, \widetilde{E}) \in \widetilde{\mathcal{G}}$ of problem (P1). Figure 8 illustrates an example. The construction and the arguments are similar to those used in the the proof of Lemma 3. The definitions of the sets $\widetilde{V}_I$, $\widetilde{V}_F$, $\widetilde{E}_j$, $1 \leq j \leq e$, are the same as above.

For each task $i_j \in \widetilde{V}_I$, we construct a chain of r_{i_j} UET tasks in G which are assigned to processor j, $1 \leq j \leq s$. In our example of Figure 8, $\widetilde{V}_I = \{\widetilde{1}, \widetilde{2}, \widetilde{3}, \widetilde{4}\}$. Thus, in G we construct 4 chains of UET tasks, chains of lengths $2, 1, 2, 1$ to be run on processors P1, P2, P3 and P4, respectively.

For each subset $\widetilde{E}_j$, $1 \leq j \leq e$, we construct a chain of $\max_{(u,v)\in\widetilde{E}_j} l(u, v)$ UET tasks which are assigned to processor $s + j$, $1 \leq j \leq e$. For the example of Figure 8, since the partition is $\widetilde{E}_1 = \{(\widetilde{2}, \widetilde{5}), (\widetilde{2}, \widetilde{6})\}$, $\widetilde{E}_2 = \{(\widetilde{6}, \widetilde{8}), (\widetilde{6}, \widetilde{9})\}$ and $\widetilde{E}_3 = \{(\widetilde{3}, \widetilde{7}), (\widetilde{4}, \widetilde{7}), (\widetilde{5}, \widetilde{7})\}$, we construct on processors (P5, P6, P7) three chains of $2, 2, 3$ UET tasks, respectively.

For each task $f_j \in \widetilde{V}_F$, we construct a chain of d_{f_j} UET tasks in G which are assigned to processor $s + e + j$, $1 \leq j \leq t$. For the example of Figure 8, we construct four chains of $3, 2, 1, 2$ UET tasks on the last four processors P8, P9, P10 and P11.

Thus, in the parallel-processor scheduling problem, there are $m = s + e + t$ processors in total, each of which executes a chain of UET tasks of G. We now



$\widetilde{G}$ G

Figure 8: Construction of graph G for problem (PR2) with blocking from an instance $\widetilde{G}$ of problem (P1).

define the communications between these chains, which result in communications among the m processors. These communications are defined according to the set of vertices in $\widetilde{V}$. We will add some more UET tasks when necessary.

Firstly, we consider only the vertices of type 0, 1, and 3 that are not the closest immediate successor (i.e. the one with the smaller precedence delay between the vertex and its predecessor) of some type 2 vertex. For each $u \in \widetilde{V}$ satisfying this property,

Case 1: if $\widetilde{P}(u) = \widetilde{S}(u) = \emptyset$, then there are j and k such that $u \equiv i_j \equiv f_k$.

We construct an arc from the r_{i_j}-th task on processor j to the first task on

processor $s+e+k$ (e.g. in Figure 8, from the second task on P1 to the first task on P8);

Case 2: if $\widetilde{P}(u)=\emptyset$ and $\widetilde{S}(u)\neq\emptyset$, then there are j and k such that $u\equiv i_j$ and for all $v\in\widetilde{S}(u)$, $(u,v)\in\widetilde{E}_k$. If $|\widetilde{S}(u)|>1$, then we construct an arc from the r_{i_j}-th task on processor j to the first task on processor $s+k$. If, however, $\widetilde{S}(u)=\{v\}$, then we construct an arc from the r_{i_j}-th task on processor j to the $\left(\max_{u'\in\widetilde{P}(v)} l_{u',v}-l_{u,v}+1\right)$-st task on processor $s+k$ (e.g. in Figure 8, from the first task on P4 to the second task on P7);

Case 3: if $\widetilde{P}(u)\neq\emptyset$ and $\widetilde{S}(u)=\emptyset$, then there are j and k such that $u\equiv f_k$, and for all $t\in\widetilde{P}(u)$, $(t,u)\in\widetilde{E}_j$. We construct an arc from the $\left(\max_{t\in\widetilde{P}(u)} l_{t,u}\right)$-th task on processor $s+j$ to the first task on processor $s+e+k$ (e.g. in Figure 8, from the third task on P7 to the first task on P11);

Case 4: if $\widetilde{P}(u)\neq\emptyset$ and $\widetilde{S}(u)\neq\emptyset$, then then there are j and k such that for all $t\in\widetilde{P}(u)$, $(t,u)\in\widetilde{E}_j$, and for all $v\in\widetilde{S}(u)$, $(u,v)\in\widetilde{E}_k$. If $|\widetilde{S}(u)|>1$, then we construct an arc from the $\left(\max_{t\in\widetilde{P}(u)} l_{t,u}\right)$-th task on processor $s+j$ to the first task on processor $s+k$. If, however, $\widetilde{S}(u)=\{v\}$, then we construct an arc from the $\left(\max_{t\in\widetilde{P}(u)} l_{t,u}\right)$-th task on processor $s+j$ to the $\left(\max_{u'\in\widetilde{P}(v)} l_{u',v}-l_{u,v}+1\right)$-st task on processor $s+k$ (e.g. in Figure 8, from the second task on P5 to the third task on P7).

We mark all these vertices for which we have already constructed the corresponding communications.

Secondly, we consider the remaining unmarked type-2 vertices of $\widetilde{V}$ which have no unmarked predecessors. Let u be such a vertex.

If $\widetilde{P}(u)=\emptyset$, then there are j and k such that $u\equiv i_j$, and for $v,w\in\widetilde{S}(v)$, $(u,v),(u,w)\in\widetilde{E}_k$. We construct an arc from the r_{i_j}-th task on processor j to the first task on processor $s+k$ (e.g. in Figure 8, from the first task on P2 to the first task on P5). If, however, $\widetilde{P}(u)\neq\emptyset$, then there are j and k such that for all $t\in\widetilde{P}(u)$,

$(t,u) \in \widetilde{E}_j$, and $v,w \in \widetilde{S}(v)$, $(u,v),(u,w) \in \widetilde{E}_k$. We construct an arc from the $\left(\max_{t\in\widetilde{P}(u)} l_{t,u}\right)$-th task on processor $s+j$ to the first task on processor $s+k$.

At the end of the second phase, we mark all vertices for which we have constructed the corresponding communications.

Thirdly, we consider the closest immediate successors (which are unmarked) of type-2 vertices which have no unmarked predecessor. Let u be such a unmarked closest immediate successor of a marked type-2 vertex t.

If $\widetilde{S}(u) = \emptyset$, then there is some k such that $u \equiv f_k$. We add a chain of $l_{t,u}$ UET tasks on the processor which is the sender of the communication corresponding to task t, and we construct an arc from the last added task on this processor to the first task on processor $s+e+k$ (e.g. in Figure 8, from the third task that we add on P2 to the first task on P9); If $\widetilde{S}(u) \neq \emptyset$, then then there is some k such that for all $v \in \widetilde{S}(u)$, $(u,v) \in \widetilde{E}_k$. We add a chain of $l_{t,u}$ UET tasks on the processor which is the sender of the communication corresponding to task t. If $|\widetilde{S}(u)| > 1$, then we construct an arc from the last added task on this processor to the first task on processor $s+k$ (e.g. in Figure 8, from the second task that we add on P2 to the first task on P6). If, however, $\widetilde{S}(u) = \{v\}$, then we construct an arc from the last added task on this processor to the $\left(\max_{u'\in\widetilde{P}(v)} l_{u',v} - l_{u,v} + 1\right)$-st task on processor $s+k$.

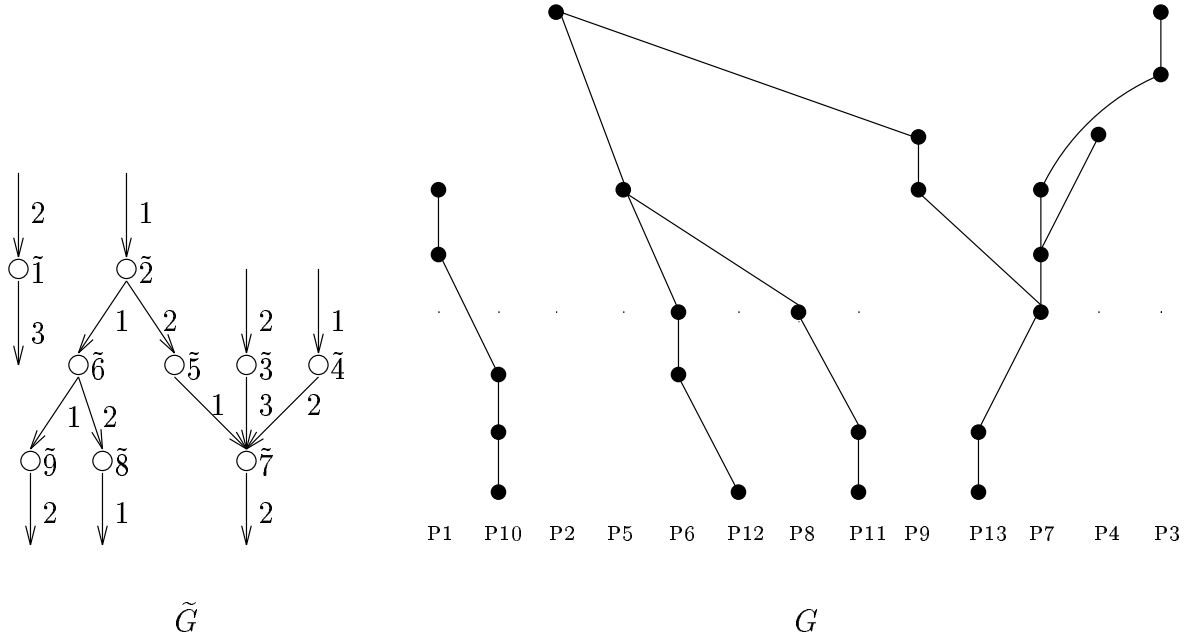
We mark vertex u, and we repeat this step (the third one) until all vertices are marked.

By the similar arguments as in the the proof of Lemma 3, we can show that any solution for the problem (P1) is a solution for the problem (PR2), and that the converse is also true. ∎

C Proof of Lemma 6

Given an instance graph $\widetilde{G} = (\widetilde{V},\widetilde{E}) \in \widetilde{\mathcal{G}}$ of problem (P1), we construct an instance graph $G = (V,E)$ for problem (PR4).

The construction of the graph and the arguments are similar to those used in the proof of Lemma 3 except for the definition of the sets of arcs beginning from the same vertex in $\widetilde{G}$. An example is illustrated in Figure 9.



$\widetilde{G}$ G

Figure 9: Construction of graph G for problem (PR4) with broadcasting communication from an instance $\widetilde{G}$ of problem (P1).

The definition of $\widetilde{V}_I$ is the same as that of Lemma 3. For each task $i_j \in \widetilde{V}_I$, we construct a chain of r_{i_j} UET tasks in G which are assigned to processor j, $1 \leq j \leq s$. In our example of Figure 9, $\widetilde{V}_I = \{\widetilde{1}, \widetilde{2}, \widetilde{3}, \widetilde{4}\}$. Thus, in G we construct 4 chains of UET tasks, with lengths $2, 1, 2, 1$, to be run on processors P1, P2, P3 and P4, respectively.

Let $\{\widetilde{E}_1, \widetilde{E}_2, \cdots, \widetilde{E}_e\}$ be a partition of the set of arcs $\widetilde{E}$ such that $(u_1, v_1) \in \widetilde{E}$ and $(u_2, v_2) \in \widetilde{E}$ belong to the same subset $\widetilde{E}_j$, $1 \leq j \leq e$, if and only if $v_1 = v_2$. Note that due to the restrictions made on the class of graphs $\widetilde{\mathcal{G}}$, the relation $v_1 = v_2$ is an equivalence relation defined on the set of arcs $\widetilde{E}$. For each subset $\widetilde{E}_j$, $1 \leq j \leq e$, we construct a chain of $\max_{(u,v)\in\widetilde{E}_j} l_{u,v}$ UET tasks which are assigned to processor

$s+j$, $1 \leq j \leq e$. For the example of Figure 9, since the partition is $\widetilde{E}_1 = \{(\widetilde{2},\widetilde{6})\}$, $\widetilde{E}_2 = \{(\widetilde{6},\widetilde{8})\}$, $\widetilde{E}_3 = \{(\widetilde{3},\widetilde{7}),(\widetilde{4},\widetilde{7}),(\widetilde{5},\widetilde{7})\}$, $\widetilde{E}_4 = \{(\widetilde{6},\widetilde{9})\}$ and $\widetilde{E}_5 = \{(\widetilde{2},\widetilde{5})\}$, we construct on processors P5, P6, P7, P8 and P9 five chains of $1,2,3,1,2$ UET tasks, respectively.

The definition of $\widetilde{V}_F$ is the same as that of Lemma 3. Let $\widetilde{V}_F = \{f_1, f_2, \cdots, f_t\}$ be the set of final tasks of $\widetilde{G}$. For each task $f_j \in \widetilde{V}_F$, we construct a chain of d_{f_j} UET tasks in G which are assigned to processor $s+e+j$, $1 \leq j \leq t$. For the example of Figure 9, on processors P10, P11, P12 and P13, we construct four chains of $3,2,1,2$ UET tasks respectively.

Thus, in the parallel-processor scheduling problem, there are $m = s+e+t$ processors in total, each of which executes a chain of UET tasks of G.

We now define the communications between these chains, which result in communications among the m processors. These communications are defined according to the set of vertices in $\widetilde{V}$. Note that a vertex u having several immediate successors corresponds to a broadcasting communication.

For each $v \in \widetilde{V}$,

Case 1: if $\widetilde{P}(v) = \widetilde{S}(v) = \emptyset$, then there are j and k such that $v \equiv i_j \equiv f_k$. We construct an arc from the r_{i_j}-th task on processor j to the first task on processor $s+e+k$ (e.g. in Figure 9, from the second task on P1 to the first task on P10);

Case 2: if $\widetilde{P}(v) = \emptyset$ and $\widetilde{S}(v) \neq \emptyset$, then there are j and $k_1, \cdots, k_{|\widetilde{S}(v)|}$ such that $v \equiv i_j$ and for $w_1, \cdots, w_{|\widetilde{S}(v)|} \in \widetilde{S}(v)$, $(v, w_l) \in \widetilde{E}_{k_l}$, $1 \leq l \leq |\widetilde{S}(v)|$. If $|\widetilde{S}(v)| > 1$, then we construct an arc from the r_{i_j}-th task on processor j to the first task on processor $s+k_l$ (e.g. in Figure 9, from the first task on P2 to the first task on P5, and also to the first task on P9). If, however, $\widetilde{S}(v) = \{w\}$, then we construct an arc from the r_{i_j}-th task on processor j to the $\left(\max_{v' \in \widetilde{P}(w)} l_{v',w} - l_{v,w} + 1\right)$-st task on processor $s+k$ (e.g. in Figure 9, from the first task on P4 to the second task on P7);

Case 3: if $\widetilde{P}(v) \neq \emptyset$ and $\widetilde{S}(v) = \emptyset$, then there are j and k such that $v \equiv f_k$, and for all $u \in \widetilde{P}(v)$, $(u,v) \in \widetilde{E}_j$. We construct an arc from the $\left(\max_{u \in \widetilde{P}(v)} l_{u,v}\right)$-th task on processor $s+j$ to the first task on processor $s+e+k$ (e.g. in Figure 9, from the third task on P7 to the first task on P13);

Case 4: if $\widetilde{P}(v) \neq \emptyset$ and $\widetilde{S}(v) \neq \emptyset$, then then there are j and $k_1, \cdots, k_{|\widetilde{S}(v)|}$ such that for all $u \in \widetilde{P}(v)$, $(u,v) \in \widetilde{E}_j$, and for $w_1, \cdots, w_{|\widetilde{S}(v)|} \in \widetilde{S}(v)$, $(v,w_l) \in \widetilde{E}_{k_l}$, $1 \leq l \leq |\widetilde{S}(v)|$. If $|\widetilde{S}(v)| > 1$, then we construct an arc from the $\left(\max_{u \in \widetilde{P}(v)} l_{u,v}\right)$-th task on processor $s+j$ to the first task on processors $s+k_l$, $1 \leq l \leq |\widetilde{S}(v)|$, (e.g. in Figure 9, from the first task on P5 to the first task on P6, and to the first task on P8 also). If, however, $\widetilde{S}(v) = \{w\}$, then we construct an arc from the $\left(\max_{u \in \widetilde{P}(v)} l_{u,v}\right)$-th task on processor $s+j$ to the $\left(\max_{v' \in \widetilde{P}(w)} l_{v',w} - l_{v,w} + 1\right)$-st task on processor $s+k$ (e.g. in Figure 9, from the second task on P9 to the third task on P7).

The construction of task graph G is thus completed. Using again the arguments of the proof of Lemma 3, one can show that any solution for the problem (P1) is a solution for the problem (PR4) with broadcasting, and that the converse is also true. ∎

References

[1] P. Chretienne, "A Polynomial Algorithm to Optimally Schedule Tasks over a Virtual Distributed System under Tree-like Precedence Constraints", *Euro. J. Oper. Res.*, **43** (1989), pp. 225-230.

[2] P. Chretienne, C. Picouleau, "Scheduling with Communication Delays: a Survey", in *Scheduling Theory and Its Applications*, (Eds. P. Chretienne et al.), J. Wiley, to appear, 1994.

[3] L. Finta, Z. Liu, "Single Machine Scheduling Subject to Precedence Delays", Rapport de Recherche INRIA, No. 2198, 1994. submitted.

[4] J. J. Hwang, "Deterministic Scheduling in Systems with Interprocessor Communication Times", Ph.D. Dissertation, Computer and Information Sciences Department, Univ. of Florida, 1987.

[5] J. A. Hoogeveen, J. K. Lenstra, B. Veltman, "Three, Four, Five, Six, or the Complexity of Scheduling with Communication Delays", Technical Report of CWI, BS-R922, 1992.

[6] C. Y. Lee, J. J. Hwang, Y. C. Chow, F. D. Anger, "Multiprocessor Scheduling with Interprocessor Delays", *Oper. Res. Letters*, Vol. 7, No. 3, pp. 141-147, 1988.

[7] Z. Liu, "A Note on Graham's Bound." *Information Processing Letters*, Vol. 36, pp. 1-5, 1990.

[8] Z. Liu, J. Labetoulle, "A Heuristic Method for Loading and Scheduling Flexible Manufacturing Systems", *Proc. of the Intern. Conf. Control 88*, London, IEE Conference Publication No.285, pp.195-200, 1988.

[9] C. Picouleau, *Etude de problèmes d'optimisation dans les systèmes distribués*, Thèse de l'Université Pierre et Marie Curie, France, 1992.

[10] C. Picouleau, "Two New NP-Complete Scheduling Problems with Communication Delays and Unlimited Number of Processors", to appear in *Disc. Appl. Math.*

[11] V. J. Rayward-Smith, "UET Scheduling with Unit Interprocessor Communication Delays", *Disc. Appl. Math.*, **18** (1987), pp. 55-71.

[12] J. D. Ullman, "NP-Complete Scheduling Problems", *Journal of Computer and System Sciences*, **10** (1975), pp. 384–393.

[13] B. Veltman, B. J. Lageweg, J. K. Lenstra, "Multiprocessor Scheduling with Communication Delays", *Parallel Computing*, **16** (1990), pp. 173-182.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399