Scheduling UET-UCT Series-Parallel Graphs on Two Processors

Lucian Finta, Zhen Liu, Ioannis Milis, Evripidis Bampis

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Scheduling UET-UCT Series-Parallel Graphs on Two Processors

Lucian Finta Zhen Liu Ioannis Milis Evripidis Bampis

N° 2566

Mai 1995

PROGRAMME 1

ISSN 0249-6399

INRIA
SOPHIA ANTIPOLIS

Scheduling UET-UCT Series-Parallel Graphs on Two Processors*

Lucian Finta Zhen Liu Ioannis Milis Evripidis Bampis

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

Projet MISTRAL

Rapport de recherche n° 2566 — Mai 1995 — 24 pages

Abstract: The scheduling of task graphs on two identical processors is considered. It is assumed that tasks have unit-execution-time, and arcs are associated with unit-communication-time delays. The problem is to assign the tasks to the two processors and schedule their execution in order to minimize the makespan. A quadratic algorithm is proposed to compute an optimal schedule for a class of series-parallel graphs, which includes in particular in-forests and out-forests.

Key-words: Scheduling, Makespan, Precedence Constraint, Series-Parallel Graphs, Complexity, Optimal Algorithm.

(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)

Ordonnancement de graphes séries-parallèles UET-UCT sur deux processeurs

Résumé : Le problème d'ordonnancement de graphes de tâches sur deux processeurs est analysé. Les temps de calcul des tâches sont unitaire, ainsi que les durées de communication associée aux arcs. Le problème est d'affecter les tâches aux processeurs et d'ordonnancer leur exécution de manière à minimiser la durée d'ordonnancement. Un algorithme optimal de complexité quadratique est proposé pour une classe de graphes séries-parallèles, classe qui inclut en particulier les arborescences.

Mots-clé : ordonnancement, durée d'ordonnancement, contrainte de précédence, graphe séries-parallèle, complexité, algorithme optimal.

1 Introduction

A notoriously difficult problem in the scheduling theory of parallel computation has been the minimization of makespan (i.e. schedule length) of set of partially ordered tasks with unit-execution-time (UET) tasks and unit-communication-time (UCT) delays on m (identical) processors [2].

A general description of the problem is the following. There are m identical processors and a set of n tasks to be run on those processors. The executions of the tasks are subject to precedence constraints (and communication delays) that are described by a weighted directed acyclic graph $G = (V, E)$, referred to as task graph, where the set of vertices V corresponds to the set of tasks and the set of arcs E the precedence constraints. The weight of task $i \in V$, denoted by p_i, is its execution time. The weight of arc $(i, j) \in E$, denoted by c_{ij}, is the communication time between tasks i and j, provided they are assigned to different processors. The communication time is considered to be negligible if two communicating tasks are assigned to the same processor. A task can start execution on a processor only if all its predecessors have completed execution and the interprocessor communications (if any) have completed.

According to the three-field notation scheme introduced in [4] and extended in [16] for scheduling problems with communication delays, such a problem can be denoted as $m \mid prec,\ c_{i,j},\ p_j \mid C_{\max}$, where $C_{\max}$ represents the makespan.

A special case of the problem is UET-UCT: $m \mid prec,\ c_{i,j} = 1,\ p_j = 1 \mid C_{\max}$, which was shown to be NP-hard in [9]. Even when the task graph is a tree, the problem remains NP-hard [15]. A list algorithm was proven optimal for interval-order task graphs [8]. For fixed $m \geq 2$, the only results reported in the literature are on trees (except the interval-order graphs). In particular an $O(n^{2(m-1)})$ optimal dynamic programming algorithm is presented in [13], while an $O(n)$ approximation algorithm computing a schedule whose length exceeds the optimum by no more than $m - 2$ units is presented in [7].

A challenging open problem is the two-processor scheduling with UET-UCT: $2 \mid prec,\ c_{i,j} = 1,\ p_j = 1 \mid C_{\max}$, for which the complexity is unknown. If, however, the tasks are a priori assigned to two processors, the problem is NP-hard for arbitrary task graphs [15, 3]. When the task graph is a tree, both results mentioned above for $m \geq 2$, [13] and [7], yields optimal polynomial solutions, while two more algorithms

were proposed in [5] and [14]. Although four algorithms are known for scheduling UET-UCT tree task graphs on two processors, no algorithm is known for more general classes of graphs.

In this paper, we provide a quadratic algorithm to compute an optimal schedule for a special class of task graphs, referred to as series-parallel-1 (SP1) graphs, denoted by $\mathcal{G}$. This class of graphs includes as particular cases the opposing forests and series-parallel-11 (see definition below) graphs.

In the next section we define the class of SP1 graphs and present some of its properties used in the paper. In Section 3 we present the (recursive) scheduling algorithm and prove its optimality and time complexity. In Section 4, we conclude with some remarks on future research.

2 Series-Parallel Graphs

All graphs considered in the paper, unless otherwise stated, are directed graphs (digraphs). The class of series-parallel graphs is known as a class for which several scheduling problems are polynomially solvable [1, 6, 11], while the same problems are NP-complete for a general graph. Here, we are interested in a subclass of series-parallel graphs which we call *series-parallel-1 graphs*.

We define the class and subclasses of series-parallel graphs using a quadruple notation, $G = (V, E, I, T)$ for a graph G, where V, E, I, T are respectively the sets of vertices, arcs, initial vertices (with no predecessor) and terminal vertices (with no successor).

Definition 1 The class of series-parallel graphs is defined as follows.

- The single vertex graph is a series-parallel graph.
- If $G_1 = (V_1, E_1, I_1, T_1)$ and $G_2 = (V_2, E_2, I_2, T_2)$ are series-parallel graphs, so are the graphs constructed by each of the following operations:

  Series composition : $G = G_1 \ \mathcal{S} \ G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup (T_1 \times I_2), I_1, T_2)$.

  Parallel composition : $G = G_1 \ \mathcal{P} \ G_2 = (V_1 \cup V_2, E_1 \cup E_2, I_1 \cup I_2, T_1 \cup T_2)$.

In the above definition, if the series composition applies only when $|T_1| = 1$ or $|I_2| = 1$, then we obtain the class $\mathcal{G}$ of *series-parallel-1* (SP1) graphs. It is clear that the class of SP1 graphs includes in-forests, out-forests and opposing forests as special cases. An example of an SP1 graph is illustrated in Figure 1(a).

Another subclass of series-parallel graphs which are frequently studied in the literature is series-parallel-11 (SP11):

Definition 2 The class of SP11 graphs is defined as follows.

- The single vertex graph is a SP11 graph.
- If $G_i = (V_i, E_i, I_i, T_i)$ are SP11 graphs, $0 \leq i \leq g+1$, so is the graph

$$G = \left(\bigcup_{i=0}^{g+1} V_i,\ \bigcup_{i=0}^{g+1} E_i \bigcup \left(T_0 \times \bigcup_{i=1}^{g} I_i \right) \bigcup \left(\bigcup_{i=1}^{g} T_i \times I_{g+1} \right),\ I_0,\ T_{g+1} \right).$$

It follows from the definition that any SP11 graph has a single initial vertex and a single terminal vertex. Hence, the class of SP11 graphs is a subclass of SP1 graphs.

The following fact will be useful.

Lemma 1 *If $G = (V, E) \in \mathcal{G}$, then $|E| \leq 2|V| - 2$.*

Proof. We prove the following inequality by induction on the number of vertices in the graph:

$$|E| \leq 2|V| - |I| - |T|,$$

which implies the assertion of the lemma.

If $|V| = 1$, then the result trivially holds. Assume there is $n \geq 2$ such that the inequality holds for all SP1 graphs with $|V| < n$. Let $G = (V, E) \in \mathcal{G}$ be such that $|V| = n$. Assume G is obtained from a series or parallel composition of SP1 graphs $G_1 = (V_1, E_1, I_1, T_1)$ and $G_2 = (V_2, E_2, I_2, T_2)$. Clearly $|V_1| < n$ and $|V_2| < n$. Then, by inductive assumption, $|E_i| \leq 2|V_i| - |I_i| - |T_i|$, $i = 1, 2$.

If the composition is parallel, then $I = I_1 \cup I_2$ and $T = T_1 \cup T_2$, so that

$$|E| = |E_1| + |E_2| \leq \sum_{i=1,2} 2|V_i| - |I_i| - |T_i| = 2|V| - |I| - |T|.$$

If the composition is series, then $E = E_1 \cup E_2 \cup (T_1 \times I_2)$, $I = I_1$ and $T = T_2$, so that

$$|E| = |E_1| + |E_2| + \max(|T_1|, |I_2|) < |T_1| + |I_2| + \sum_{i=1,2} 2|V_i| - |I_i| - |T_i| = 2|V| - |I| - |T|,$$

where the first equality comes from the fact that $|T_1| = 1$ or $|I_2| = 1$, and the inequality comes from the fact that for all positive numbers x and y, $\max(x, y) \leq x + y$.

Thus, by induction, the result holds for all SP1 graphs. ∎

The composition of a series-parallel graph G can be represented by a binary tree $\mathcal{T}$, referred to as *decomposition tree*. Each leaf of $\mathcal{T}$ represents a vertex in G; each internal node is labeled $\mathcal{S}$ or $\mathcal{P}$ and represents the series or parallel composition of the series-parallel-1 subgraphs which are in turn represented by the subtrees rooted at the children of the node. By convention, we assume that in the series compositions the left child precedes the right one. Thus, the decomposition tree is ordered. However the order between children of a parallel composition has no importance. Clearly, decomposition trees are not unique, as it is possible to have ties between successive compositions of the same type.

In [12] a linear time algorithm was presented for recognizing the general class of series-parallel graphs and constructing a binary decomposition tree. However, their algorithm breaks ties between successive compositions of the same type in an arbitrary way. Another algorithm was presented in [10] for the construction of a decomposition tree of a given SP1 graph. Although the algorithm applies only to the subclass of SP1 graphs, it breaks ties between successive series compositions in a particular way which is convenient to our scheduling algorithm (ties between parallel compositions does not affect our algorithm).

More specifically, we require that the series composition of G is such that G_1 is *minimal*, in the sense that there is no other series decomposition of G into G'_1 and G'_2 such that G'_1 is a proper subgraph of G_1. Such a decomposition tree for the SP1 graph of Figure 1(a) is illustrated in Figure 1(b).

We present here a new algorithm for the recognition of SP1 graphs and the construction of decomposition trees with minimal series decomposition. Our algorithm has a smaller time complexity than that of [10].

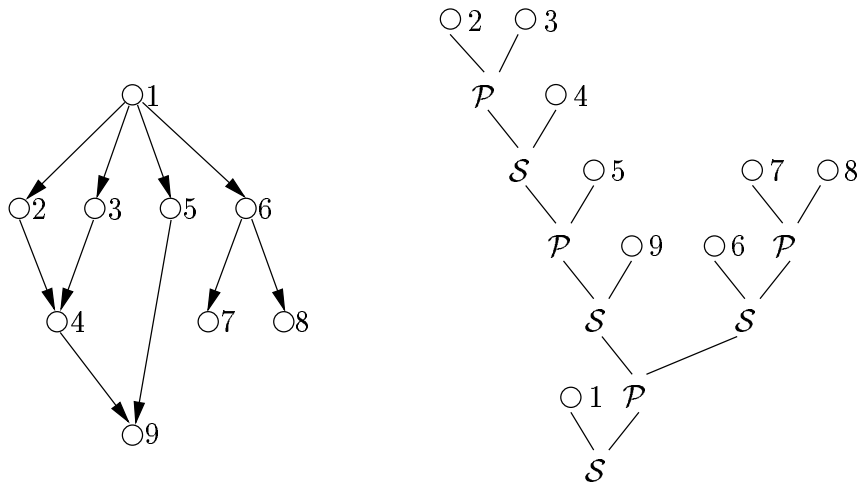

Figure 1: (a) Example of an SP1 graph and (b) the corresponding decomposition tree.

For each vertex $v \in V$ we define the following sets. $A(v)$ is the set of all predecessors (ancestors) of v, $D(v)$ is the the set of all successors (descendants) of v, including v itself, and $S(v)$ is the set of the immediate successors of v. Let $a(v) = |A(v)|$, $\vec{a} = (a(v),\ v \in V)$, and $d(v) = |D(v)|$, $\vec{d} = (d(v),\ v \in V)$.

Our algorithm will use the following facts in order to recognize and construct the decomposition tree of $G = (V, E)$.

- If $|I| = 1$, then the single vertex in I is in series composition with the remaining graph.
- If $|I| > 1$ and G is not connected, then it is composed in parallel.
- If $|I| > 1$ and G is connected, either it is composed in series or it is not an SP1 graph.

When $|I| > 1$, in order for G to be an SP1 graph provided by a series composition of G_1 and G_2, G_2 must have a single initial vertex. Let y be such a vertex. It then satisfies the equality $A(y) \cup D(y) = V$. Since the sets $A(y)$ and $D(y)$ are disjoint, we obtain

$$a(y) + d(y) = |V|.$$

If such a vertex does not exist, then clearly G does not belong to the class of SP1 graphs.

When $|I| > 1$ and G is connected, there may be several vertices y satisfying $a(y) + d(y) = |V|$. Since G is acyclic, a topologically sorted numbering of its vertices can be provided so that $u \in A(v)$ implies that $u < v$. Thus, the minimality of series decomposition is guaranteed by choosing y to be the smallest vertex satisfying the equality above. This way of series decomposition gives a priority to successive series decompositions (top-down in the initial graph) which will be crucial for our scheduling algorithm.

Our algorithm, referred to as **Decomp**, is formally summarized below in a recursive manner. Before the algorithm is called, the following preprocessing is performed on the graph $G = (V, E)$, where the variables $S(v)$ are global variables in the recursive algorithm, while the others are local ones.

- Relabel the vertices in accordance with the partial order of the graph.
- Compute the set I of initial vertices.
- For each $v \in V$ compute $a(v)$, $d(v)$ and $S(v)$.

In the algorithm **Decomp**, we denote by $R(T)$ the root of a tree. When $G' = (V', E')$ is a subgraph of $G = (V, E)$, we assume that E' is a restriction of E on the vertices of V'. Similarly, $\vec{a}(V')$ and $\vec{d}(V')$ denote the restrictions of the vectors $\vec{a}$ and $\vec{d}$ to the vertices of V'.

```
Procedure Decomp (G, I, ā, d̄);        {returns the tree T(G) }
begin
    If |V| = 1  (V = {x}) then T(G) := x
    else  If |I| = 1  (I = {x}) then
        V_2 := V − {x}; I_2 := S(x);
        For each v ∈ V_2 do a(v) := a(v) − 1;
        R(T(G)) := S;
        left-child:=x;
        right-child:=Decomp(G_2, I_2, ā(V_2), d̄(V_2))
    else
        Find the connected components, G^1, G^2, ... of G;
        If G is not connected then
            G_1 := G^1;  V_2 := V − V_1; I_1 := I − V_2;  I_2 := I − I_1;
            R(T(G)) := P;
            left-child:=Decomp(G_1, I_1, ā(V_1), d̄(V_1));
            right-child:=Decomp(G_2, I_2, ā(V_2), d̄(V_2))
        else  {i.e. G is connected }
            Find the smallest vertex satisfying a(y) + d(y) = |V|;
            If such y exists then
                V_1 := {v ∈ V | v < y}; V_2 := {v ∈ V | v ≥ y};
                I_1 := I; I_2 := {y};
                For each v ∈ V_1 do d(v) := d(v) − d(y);
                For each v ∈ V_2 do a(v) := a(v) − a(y);
                R(T(G)) := S;
                left-child:=Decomp(G_1, I_1, ā(V_1), d̄(V_1));
                right-child:=Decomp(G_2, I_2, ā(V_2), d̄(V_2))
            else return  G ∉ G
end
```

Lemma 2 *The procedure* **Decomp** *recognizes whether a given graph* $G = (V, E)$ *is SP1 and if it is, then constructs the decomposition tree in* $O(n^2)$ *time, where* $n = |V|$.

Proof. The procedure **Decomp** is called recursively $O(n)$ times. In each call $O(n)$ time is enough for the computations, even for finding the connected components, since by Lemma 1, $|E| \leq 2|V| - 2$, that is $|E| = O(n)$. The complexity of the algorithm is therefore $O(n^2)$. Since the preprocessing step can also be implemented in $O(n^2)$ time units, the complexity of the whole recognition/decomposition algorithm is $O(n^2)$. ∎

3 Scheduling

3.1 Preliminaries

We define a schedule as a function $\sigma : V \to I\!N_+ \times \{1, 2\}$, i.e. $\sigma(u) = (t_u, p_u)$ where t_u is the time slot and p_u the processor on which task u is scheduled. A schedule is *feasible* if:

- for all $u, v \in V$, $u \neq v$ implies $(t_u, p_u) \neq (t_v, p_v)$;
- if $(u, v) \in E$ then $t_u + 1 + \mathbf{1}(p_u \neq p_v) \leq t_v$,

where $\mathbf{1}(\bullet)$ is the indicator function.

The reverse function $\sigma^{-1}(t, p)$ gives the task scheduled on processor p in time slot t. In what follows we say that a schedule σ has an *idle* in time slot t if one of the processors is idle during this time slot. In this case we consider as if the idle processor is executing a fictive task, labeled 0. Processors are denoted by P1 and P2. By M we denote the makespan (length) of a schedule, that is the last time slot some task is executed on any processor:

$$M = \max\{\ t \mid \sigma^{-1}(t, 1) \neq 0 \text{ or } \sigma^{-1}(t, 2) \neq 0\}.$$

An idle in the time slot 1 (resp. M) of σ is called *left idle* (resp. *right idle*). Both left and right idles are called *extremal idles*; other idles in a schedule are called *internal idles*.

Roughly speaking the idea of the algorithm is to combine recursively the schedules of the subgraphs into which G is decomposed, following its decomposition tree. In a series composition ($G = G_1\ \mathcal{S}\ G_2$), it is clear that all tasks of G_1 are predecessors of all tasks of G_2 and therefore the schedule of G will be a concatenation of the schedules of G_1 and G_2. In a parallel composition ($G = G_1\ \mathcal{P}\ G_2$) there is no precedence relation between tasks of G_1 and tasks of G_2. The idea is to "merge" the schedules of G_1 and G_2 by, in general, using the schedule of one graph and then filling-up the idles of this schedule with the tasks of the second graph, executed sequentially.

In what follows, we will consider a subgraph of an SP1 graph in its decomposition tree. In particular, we shall construct schedules of the subgraph by taking into

account the next (series or parallel) composition to be operated on it, i.e. its father in the decomposition tree. For all $G \in \mathcal{G}$, the pair $(G, \mathcal{O})$, where $\mathcal{O} \in \{\mathcal{S}^+, \mathcal{S}^-, \mathcal{P}\}$, denotes the type of next composition ($\mathcal{S}$ or $\mathcal{P}$) to which graph G participates. The sign denotes that G is a right (+) or left (-) child in a series composition. In a parallel composition interchanging children does not make any difference.

By the definition of the series composition ($G = G_1 \ \mathcal{S} \ G_2$) either $|T_1| = 1$ or $|I_2| = 1$ and therefore either σ_1 has a right idle (if $|T_1| = 1$) or σ_2 has a left idle (if $|I_2| = 1$). These idles are the key point in the development of our scheduling algorithm.

Definition 3 A schedule σ of G is said to be *nice* with respect to $(G, \mathcal{O})$ if all the following hold:

(i) In each time slot, at least one processor is busy: $\sigma^{-1}(t,1) + \sigma^{-1}(t,2) \neq 0$, $1 \leq t \leq M$.

(ii) In any two consecutive time slots P1 cannot be idle in the first if P2 is idle in the second or vice-versa, i.e. $\sigma^{-1}(t,i) + \sigma^{-1}(t+1, (i \bmod 2)+1) \neq 0$, $i = 1, 2$.

(iii) If $\mathcal{O} = \mathcal{P}$, then σ is an *optimal* schedule of G with the most possible extremal idles. Moreover,

- if σ has only one extremal idle and at least one internal idle then there is no optimal schedule with the extremal idle in the opposite side, and
- if σ has no extremal idle and at least one internal idle then there is no schedule of length $M+1$ with two extremal idles.

(iv) If $\mathcal{O} = \mathcal{S}^-$ (`resp.` $\mathcal{S}^+$), then σ is the *shortest* schedule with a right (`resp. left`) idle, and if possible, a left (`resp. right`) idle, i.e. σ either is optimal or has the length of the optimal schedule plus one provided that there is no optimal schedule with a right (`resp. left`) idle.

Note that the shortest schedule in (iv) may not be an optimal schedule as we required an extremal idle. In our scheduling algorithm, we will recursively compute nice schedules and combine these schedules according to the composition operations. To this end, we will use often two symmetric operations **stretch-right** and **stretch-left**. Given a schedule σ of length M satisfying properties (i) and (ii) of Definition 3, each of these operations increases the length M to $M+l$, provided that $M+l \leq |V|$, by stretching σ right (resp. left) and preserving the same properties.

Lemma 3 *Every schedule σ of a graph $G = (V, E)$ with length $M < |V|$, satisfying properties (i) and (ii) of Definition 3, can be stretched to a schedule of length $M + l$, where $M + l \leq |V|$, preserving the same properties. These stretch operations can be done in $O(|V|)$ time.*

Proof. We prove the lemma for the stretch-right operation, the stretch-left being symmetric.

Consider first the case $l = 1$. Since $M < |V|$, there is at least one time slot where both processors are busy. Let t be the last time slot where both processors are busy. By properties (i) and (ii) of Definition 3, one processor is always idle and the other always busy after time slot t. Assume without loss of generality that processor P1 is always busy after time t.

Let u, v be the tasks executed on P1 and P2 in time slot t. The stretched schedule is identical to σ until time slot $t - 1$. If P1 is idle in the time slot $t - 1$, then tasks u, v are executed on P2 in the time slots t and $t + 1$ respectively. All the other tasks executed on P1 during the time slots $t + 1, t + 2, \cdots, M$, are moved to P2 and they are executed during the time slots $t + 2, t + 3, \cdots, M + 1$. Otherwise, if P1 is not idle in time slot $t - 1$, we execute u, v on P1 in time slots t and $t + 1$ respectively. All the other tasks on P1 are shifted right one time slot.

The resulting schedule is clearly feasible and satisfies properties (i) and (ii) of Definition 3. The time complexity of this operation is $O(n)$.

Consider now $l > 1$. Since $M + l \leq |V|$, there are at least l time slot where both processors are busy. Let $t_1 < t_2 < \cdots < t_l$ be the right-most time slots where both processors are busy. Then, we start the above rearrangement and shift operations from t_1. The tasks scheduled after t_i and before t_{i+1} are shifted right i time slots, $i = 1, 2, \ldots, l$, where t_{i+1} is defined by convention as $M + 1$. Again the time complexity of the whole operation is $O(n)$. ∎

In the next two subsections we compute a nice schedule of a graph G given the nice schedules σ_1 and σ_2 of the graphs G_1 and G_2 to which G is decomposed according the decomposition tree $\mathcal{T}$. Following Definition 2, it is clear that in order to decide about the nice schedule of a composition we have to take into account the next composition operation to which G participates.

3.2 Schedule of a parallel composition

Consider first the case $G = G_1 \mathcal{P} G_2$, where $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Let σ_1 (resp. σ_2) be a nice schedule of G_1 (resp. G_2) with respect to $(G_1, \mathcal{P})$ (resp. $(G_2, \mathcal{P})$) which results in a makespan of M_1 (resp. M_2). Let $M' = \left\lceil \frac{|V_1|+|V_2|}{2} \right\rceil$, where $\lceil x \rceil$ denotes the smallest integer greater than or equal to x.

Since G_1 and G_2 participate in a parallel composition we can interchange the indices without loss of generality. Thus, we assume that $|V_1| \geq |V_2|$. Note that in this case, $M_2 \leq M'$ (otherwise, $|V_1| \geq |V_2| \geq M_2 \geq M' + 1$ so that $|V_1| + |V_2| \geq 2M' + 2$, which is a contradiction with the definition of M').

The algorithm **SchPar** below constructs a nice schedule σ for the graph G with respect to $(G, \mathcal{O})$, where $\mathcal{O} \in \{\mathcal{S}^-, \mathcal{S}^+, \mathcal{P}\}$. Note that the input arguments σ_1 and σ_2 allow one to compute the set of vertices V_1 and V_2 without having knowledge on the graphs G_1 and G_2.

procedure SchPar $(\sigma_1, \sigma_2, \mathcal{O})$; $\{$*returns a nice schedule* σ.$\}$
begin
If $\mathcal{O} = \mathcal{S}^-$ (`resp.` $\mathcal{S}^+$)
  If $|V_1| = |V_2|$ **then**
    If $|V_1| = |V_2| = 1$ **then** schedule V_1 and V_2 sequentially on P1
    else $\{$*i.e.* $|V_1| = |V_2| > 1\}$
      Schedule G_1 sequentially on P1 in time slots $1, 2, ..., M'$ and
        G_2 on P2 in the time slots $2, 3, ..., M' + 1$
  else $\{$*i.e.* $|V_1| > |V_2|\}$
    If σ_1 has no right (`resp. left`) idle **then**
      stretch-right (`resp. stretch-left`) σ_1 to $M_1 + 1$, $M_1 := M_1 + 1$;
    If $M' \geq M_1$ **then**
      If $|V_1| + |V_2|$ is even **then**
        stretch-left (`resp. stretch-right`) σ_1 to length $M' + 1$; $M_1 := M' + 1$;
      else $\{$*i.e.* $|V_1| + |V_2|$ *is odd* $\}$
        If $M' > M_1$ **then**
          stretch-left (`resp. stretch-right`) σ_1 to length M'; $M_1 := M'$;
    Fill-up backwards (`resp. forwards`) the idles of σ_1 with tasks of V_2
      starting from the $(M_1 - 1)^{th}$ (`resp.` 2^{nd}) time slot ;
else $\{$*i.e.* $\mathcal{O} = \mathcal{P}\}$
  If $|V_1| = |V_2|$ **then** Schedule sequentially G_1 on P1 and G_2 on P2;
  else $\{$*i.e.* $|V_1| > |V_2|\}$
    If $M' > M_1$ **then** stretch-left σ_1 to length M'; $M_1 := M'$;
    If $|V_1| + |V_2|$ is odd **or** $M_1 > M'$ **then**
      Fill-up forwards the idles of σ_1 with tasks of V_2 starting from the 2^{nd} time slot
    else $\{$*i.e.* $|V_1| + |V_2|$ *is even and* $M_1 \leq M'\}$
      Fill-up forwards the idles of σ_1 with tasks of V_2 starting from the 1^{st} time slot
end

Lemma 4 *Given the schedules* σ_1, σ_2 *of task graphs* $G_1, G_2 \in \mathcal{G}$ *which are nice with respect to* $(G_1, \mathcal{P})$ *and* $(G_2, \mathcal{P})$, *the algorithm SchPar computes a nice schedule* σ *of* $G = G_1 \mathcal{P} G_2$ *with respect to* $(G, \mathcal{O})$, $\mathcal{O} \in \{\mathcal{S}^-, \mathcal{S}^+, \mathcal{P}\}$, *in* $O(|V|)$ *time. The length* M *of* σ *is given as follows:*

- *If* $\mathcal{O} = \mathcal{P}$, *then* $M = \max(M_1, M')$.
- *If* $\mathcal{O} = \mathcal{S}^-$ (`resp.` $\mathcal{S}^+$), *then*

$$M = \left\lceil \max\left(M_1, \frac{|V_1| + |V_2|}{2}\right) + \frac{\mathbf{1}(\sigma_1 \text{ has a right (resp. left) idle})}{2} \right\rceil .$$

Proof. Note first that unless in trivial cases, the schedule σ constructed in algorithm SchPar is based on σ_1. Since any feasible schedule of G has length at least $\max(M_1, M')$, we start with an intermediate schedule σ' which is identical to σ_1 (if $M_1 \geq M'$) or σ_1 stretched to length M' (if $M_1 < M'$). When $M_1 \leq M'$, the number of idles in σ' is

$$2M' - |V_1| = \begin{cases} |V_2| + 1, & \text{if } |V_1| + |V_2| \text{ is odd} \\ |V_2|, & \text{if } |V_1| + |V_2| \text{ is even.} \end{cases}$$

When $M_1 > M'$, the number of idles is at least $|V_2|+2$. Since the schedule σ' satisfies properties (i) and (ii) of Definition 3, all tasks V_2 can be scheduled sequentially in these idles. Thus σ' is an optimal schedule of G.

However, as we are searching for a nice schedule of $(G, \mathcal{O})$, we have to fill up the idles in particular way, and stretch further this schedule when necessary.

Consider first the case $\mathcal{O} = \mathcal{P}$. The algorithm SchPar clearly provides an optimal schedule of length $M = \max(M_1, M')$ with the most possible extreme idles.

Moreover, algorithm SchPar provides a nice schedule of $(G, \mathcal{P})$. Indeed, if $|V_1| + |V_2|$ is even and $M_1 \leq M'$ (which includes the case $|V_1| = |V_2|$), no optimal schedule allows idle. Thus, σ is optimal and nice schedule of G with respect to $(G, \mathcal{P})$.

If $M_1 \leq M'$ and $|V_1| + |V_2|$ is odd, i.e. $|V_1| + |V_2| = 2M' - 1$, then there is exactly one idle in any schedule with length M'. Consider schedule σ' with length M' which is identical to σ_1 or obtained after a stretch-left operation from σ_1. There are two subcases: (i) σ' has at least one extremal idle, (ii) σ' has no extremal idle. In case (i), schedule σ preserves either the left or the right extremal idle of σ'. In case (ii), no stretch operation is performed on σ_1 to obtain σ' (otherwise extremal idles would be created). Since σ_1 is a nice schedule of G_1 (which implies that there is no feasible schedule of G_1 with length $M' + 1$ and two extremal idles), σ constructed by the algorithm is optimal and nice with one internal idle.

If $M_1 > M'$ it is easy that the total number of idles of σ_1 is at least $|V_2| + 2$, thus there is at least one internal idle in σ_1. It is clear that the schedule σ of G constructed by the algorithm uses no extremal idles (if any) of σ_1. Thus, the niceness of σ_1 with respect to $(G_1, \mathcal{P})$ guarantees the niceness of σ with respect to $(G, \mathcal{P})$. More precisely, if the optimal nice σ_1 has two extremal idles, it is clear that schedule σ of G is of length M_1 with two extremal idles, and is therefore nice. If σ_1 has only one extremal idle and since it is nice, it follows that there is no optimal

schedule with one idle in the opposite side neither for G_1 nor for G. In this case σ constructed by the algorithm is optimal and nice with one extremal idle and at least one internal idle. If the optimal nice σ_1 has no extremal idle, one can easily see from the niceness of σ_1 with respect to $(G_1, \mathcal{P})$ that there is no feasible schedule neither for G_1 nor for G with length M_1 with some extremal idle, and there is no feasible schedule neither for G_1 nor for G with length $M_1 + 1$ with two extremal idles. Note however that in this case σ has at least two internal idles.

Consider now the case $\mathcal{O} = \mathcal{S}^-$ (`resp.` $\mathcal{S}^+$). In order for σ to be nice it must have a right (`resp.` `left`) idle. When $|V_1| = |V_2|$, it is simple to see that the schedule σ has the required properties. Assume now $|V_1| > |V_2|$.

If $M_1 > M'$ then σ_1 has at least $|V_2|+2$ idles, thus at least one internal idle. Since σ_1 is optimal for G_1, there is no feasible schedule for $G_1 \mathcal{P} G_2$ with length strictly less than M_1. Consider the following subcases.

- If σ_1 has two extremal idles then schedule σ constructed by the algorithm has length M_1 and two extremal idles, and is therefore optimal for G and nice with respect to $(G, \mathcal{S}^-)$ (`resp.` $(G, \mathcal{S}^+)$).
- If σ_1 has a right (`resp.` `left`) idle, but not the opposite one, then, since σ_1 is nice with respect to $(G_1, \mathcal{P})$, it is clear that there is no feasible schedule of length M_1 for G with neither two extremal idles nor left (`resp.` `right`) idle and no idle in the opposite side. Thus, the algorithm constructs an optimal schedule of length M_1 for G with a right (`resp.` `left`) idle and at least one internal idle, which is therefore nice with respect to $(G, \mathcal{S}^-)$ (`resp.` $(G, \mathcal{S}^+)$).
- If σ_1 has a left (`resp.` `right`) idle, but not the opposite (needed) one, since σ_1 is nice with respect to $(G_1, \mathcal{P})$, it is clear that there is no feasible schedule of length M_1 for G with a right (`resp.` `left`) idle. In this case the algorithm constructs a nice schedule σ of length $M_1 + 1$ with two extremal idles.
- If σ_1 has no extremal idles, then, a similar argument as in the previous case implies that the algorithm constructs a nice schedule of length $M_1 + 1$ with only one (the needed one) extremal idle and at least one internal idle.

If $M_1 = M'$ then σ_1 has at most $|V_2|+1$ idles, thus any schedule of G with length M' (if any) cannot have more then one idle. If $|V_1| + |V_2|$ is odd, then σ_1 has exactly $|V_2| + 1$ idles. Consider the following three subcases.

- If σ_1 has a right (`resp. left`) idle, i.e. either it has two extremal idles or right (`resp. left`) idle and no left (`resp. right`) one, the algorithm constructs an optimal and nice schedule of G with one right (`resp. left`) idle.
- If σ_1 has left (`resp. right`) idle and no right (`resp. left`) one, since σ_1 is nice then there is no feasible schedule of G with length M' and one right (`resp. left`) idle. Schedule σ constructed by the algorithm has length M_1+1 with two extremal idles, and is therefore nice with respect to $(G, \mathcal{S}^-)$ (`resp.` $(G, \mathcal{S}^+)$).
- If σ_1 has no extremal idles, since it is nice, then it is clear that there is no feasible schedule of length M_1+1 for G with two extremal idles. The algorithm constructs a schedule of length $M_1 + 1$ with the needed extremal idle.

If, however, $|V_1|+|V_2|$ is even, i.e. $|V_1|+|V_2| = 2M'$, then any schedule of G of length M' contains no idle. Thus any nice schedule of G with respect to $(G, \mathcal{S}^-)$ (`resp.` $(G, \mathcal{S}^+)$) must have length at least $M' + 1$. If σ_1 has at least one extremal idle, the algorithm constructs such a nice schedule σ with two extremal idles. If σ_1 has no extremal idle, since it has internal idles and is nice with respect to $(G_1, \mathcal{P})$, there is no schedule of length $M' + 1$ with two extremal idles either for G_1 or for G. The algorithm constructs a nice schedule σ of length $M' + 1$ with a right (`resp. left`) idle and no idle in the opposite side.

If $M_1 < M'$. consider first the case when $|V_1| + |V_2|$ is odd. Then any schedule of length M' have exactly one idle. Our algorithm constructs an optimal and nice schedule σ of length M' with one right (`resp. left`) idle and no idle in the opposite side. If $|V_1| + |V_2|$ is even, i.e. $|V_1| + |V_2| = 2M'$, then there is no schedule of G of length M' containing some idles. The algorithm constructs a nice schedule σ of length $M' + 1$ with two extremal idles.

Finally, consider the time complexity of procedure SchPar. It depends on the stretching and filling-up operations executed. Both operations can easily implemented in $O(n)$ time by a simple traversal of the schedules involved. Thus, the algorithm SchPar has linear time complexity. ∎

3.3 Schedule of a series composition

Consider now the case $G = G_1 \mathcal{S} G_2$, where $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Again, let σ_1 (resp. σ_2) be a nice schedule of G_1 (resp. G_2) with respect to $(G_1, \mathcal{S}^-)$ (resp. $(G_2, \mathcal{S}^+)$) which results in a makespan of M_1 (resp. M_2).

By the niceness assumption of σ_1 and σ_2, σ_1 has a right idle and σ_2 a left one. We suppose that the right idle in σ_1 and the left idle on σ_2 are in the same processor (otherwise a renumbering of the processors in one of schedules yields the desired property).

The algorithm **SchSer** below constructs a nice schedule σ of G with respect to $(G, \mathcal{O})$. Note that our decomposition tree provides minimal series composition. Thus we only need to consider the cases $\mathcal{O} \in \{\mathcal{P}, \mathcal{S}^+\}$.

procedure SchSer $(\sigma_1, \sigma_2, \mathcal{O})$; {*returns a nice schedule* σ.}
begin
σ is the concatenation of σ_1 and σ_2;
If $\mathcal{O} = \mathcal{S}^+$ and σ_1 has no left idle **then** stretch-left σ;
end

Lemma 5 *Given the schedules σ_1, σ_2 of task graphs $G_1, G_2 \in \mathcal{G}$ which are nice with respect to $(G_1, \mathcal{S}^-)$ and $(G_2, \mathcal{S}^+)$, the algorithm SchSer computes a nice schedule σ of $G = G_1 \mathcal{S} G_2$ with respect to $(G, \mathcal{O})$, $\mathcal{O} \in \{\mathcal{S}^+, \mathcal{P}\}$, in $O(n)$ time, where $n = |V|$. The length M of σ is given as follows:*

- *If $\mathcal{O} = \mathcal{P}$, then $M = M_1 + M_2$.*
- *If $\mathcal{O} = \mathcal{S}^+$, then $M = M_1 + M_2 + \mathbf{1}(\sigma_1$ has a left idle$)$.*

Proof. In a series composition ($G = G_1 \mathcal{S} G_2$) all tasks of G_1 are predecessors of all tasks of G_2. Thus any schedule of graph G contains at least two internal idles. Since σ_1 and σ_2 are nice, σ_1 has a right idle and σ_2 a left one. Note also that at least one of the schedules σ_1 and σ_2 is optimal due to the fact that either T_1 or I_2 is singleton. We will show that the concatenation of σ_1 and σ_2 yields an optimal schedule of G.

Consider first the case $\mathcal{O} = \mathcal{P}$. If both schedules σ_1, σ_2 are optimal, then clearly the concatenation is optimal. We show below that the simple concatenation also yields a nice schedule.

- If σ_1 and σ_2 have both two extremal idles then clearly σ has also two extremal idles and thus is a nice schedule of G.
- If σ_1 has two extremal idles and σ_2 only one (which is necessarily the left idle), since σ_2 is nice with respect to $(G_2, \mathcal{S}^+)$, we can conclude that there is no optimal schedule with length $M_1 + M_2$ for G with one right idle (otherwise there is a subschedule in σ for G_2 with two extremal idles and length M_2).
- If σ_2 has two extremal idles and σ_1 only one (which is necessarily the right idle), by similar arguments one can conclude that there is no optimal schedule for G with one left idle.
- When both schedules σ_1, σ_2 have only one extremal idle, since σ_1 (resp. σ_2) is nice with respect to $(G_1, \mathcal{S}^-)$ (resp. $(G_2, \mathcal{S}^+)$), it is clear that there is no optimal schedule for G_1 (resp. G_2) with two extremal idles, thus there is no schedule for G of length $M_1 + M_2 + 1$ with two extremal idles (otherwise there exists a subschedule in σ either for G_1 of length M_1, or for G_2 of length M_2 with two extremal idles).

If only one of the schedules σ_1 and σ_2 is optimal, we consider the case σ_1 is suboptimal. The case that σ_2 is suboptimal can be tackled in an analogous way. Owing to the fact that σ_1 is a nice schedule with respect to (G_1, S^-), it follows that there is no optimal schedule for G_1 with one right idle. Thus all optimal schedules of G_1 which have length $M_1 - 1$ should end with both processors busy in the last time slot. Due to the interprocessor communications between the terminal tasks of G_1 and the initial task of G_2, the concatenation of any optimal schedule of G_1 with an optimal schedule of G_2 (e.g. σ_2) results in an idle at time slot M_1 so that the total optimal schedule of G has length $M_1 + M_2$. Since the concatenation of nice schedules σ_1 and σ_2 yields the same makespan, this concatenation is also optimal for G. Using now similar arguments as in the previous case, one concludes that this simple concatenation of nice schedules yields also a nice schedule with respect to $(G, \mathcal{P})$.

Consider now the case $\mathcal{O} = \mathcal{S}^+$. In the algorithm, we first construct σ as before with length $M = M_1 + M_2$. This schedule is nice with respect to $(G, \mathcal{S}^+)$ whenever

σ_1 has a left idle. When σ_1 does not have a left idle, we make a stretch-left operation on σ to create a left idle on σ. In order to prove the resulted schedule is nice, we consider two subcases.

- If σ has a right idle, one knows that there is no optimal schedule with a left idle from the $\mathcal{P}$ case. Thus any schedule with left idle should have length $M_1 + M_2 + 1$. Then it is enough to stretch-left σ and the resulted schedule will be nice (with two extremal idles) of length $M = M_1 + M_2 + 1$.
- If σ has no right idle, one knows that there is no schedule of length $M_1 + M_2 + 1$ with two extremal idles from the $\mathcal{P}$ case. Since the next composition is $\mathcal{S}^+$ the shortest schedule with one left idle must have length $M = M_1 + M_2 + 1$. It is clear that the required left idle can be created by stretch-left σ.

The complexity of the SchSer procedure is $O(n)$ since only a traversal of the schedules and possibly a stretch operation are needed. ∎

3.4 Main Result

Given the decomposition tree $\mathcal{T}(G)$ of a series-parallel graph G, we recursively apply algorithms SchPar and SchSer to provide nice schedules of subgraphs of G, and finally an optimal schedule of G. This procedure is described in the algorithm **Sched** below. The optimal schedule of G is obtained using the arguments $\mathcal{T}(G), \mathcal{P}$, i.e., **Sched**$(\mathcal{T}(G), \mathcal{P})$.

Algorithm Sched $(\mathcal{T}, \mathcal{O})$; {*returns a nice schedule of an SP1 graph represented by* $\mathcal{T}$.}
begin
  If $R(\mathcal{T}) \in V$ **then return** the optimal schedule of the single task on P1;
  else If $R(\mathcal{T}) = \mathcal{S}$ **then**
    $\sigma_1 :=$ **Sched**(*left-child,* $\mathcal{S}^-$*);*
    $\sigma_2 :=$ **Sched**(*right-child,* $\mathcal{S}^+$*);*
    $\sigma :=$ **SchSer**$(\sigma_1, \sigma_2, \mathcal{O})$;
  else {*i.e.* $R(\mathcal{T}) = \mathcal{P}$}
    $\sigma_1 :=$ **Sched**(*left-child,* $\mathcal{P}$*);*
    $\sigma_2 :=$ **Sched**(*right-child,* $\mathcal{P}$*);*
    $\sigma :=$ **SchPar**$(\sigma_1, \sigma_2, \mathcal{O})$;
end.

We prove the following theorem.

Theorem 1 *For every UET-UCT series-parallel-1 task graph* G *with* n *tasks, there is an* $O(n^2)$ *algorithm to find an optimal schedule of* G *on two processors.*

Proof. It is easily seen, and also simply shown by induction on the number of tasks of an SP1 task graph, that the algorithm **Sched** provides nice schedules. As a nice schedule of a graph with respect to a parallel composition $\mathcal{P}$ is an optimal schedule of the graph, we conclude that **Sched**$(\mathcal{T}(G), \mathcal{P})$ provides an optimal schedule of G.

Since the time complexities of the procedures SchPar and SchSer are linear, and since there are at most n compositions in $\mathcal{T}(G)$, the time complexity of the algorithm **Sched** is clearly $O(n^2)$.

Given also that the time complexity of the recognition/decomposition algorithm **Decomp** is also $O(n^2)$, it follows that the complexity of the whole problem remains $O(n^2)$. ∎

We complete this section with an illustration of the algorithms. In Figure 2, we present some intermediate nice schedules and the final optimal schedule for the task graph of Figure 1(a). Note that the dashed part in the schedule of (g) is the stretched schedule of that of (d).

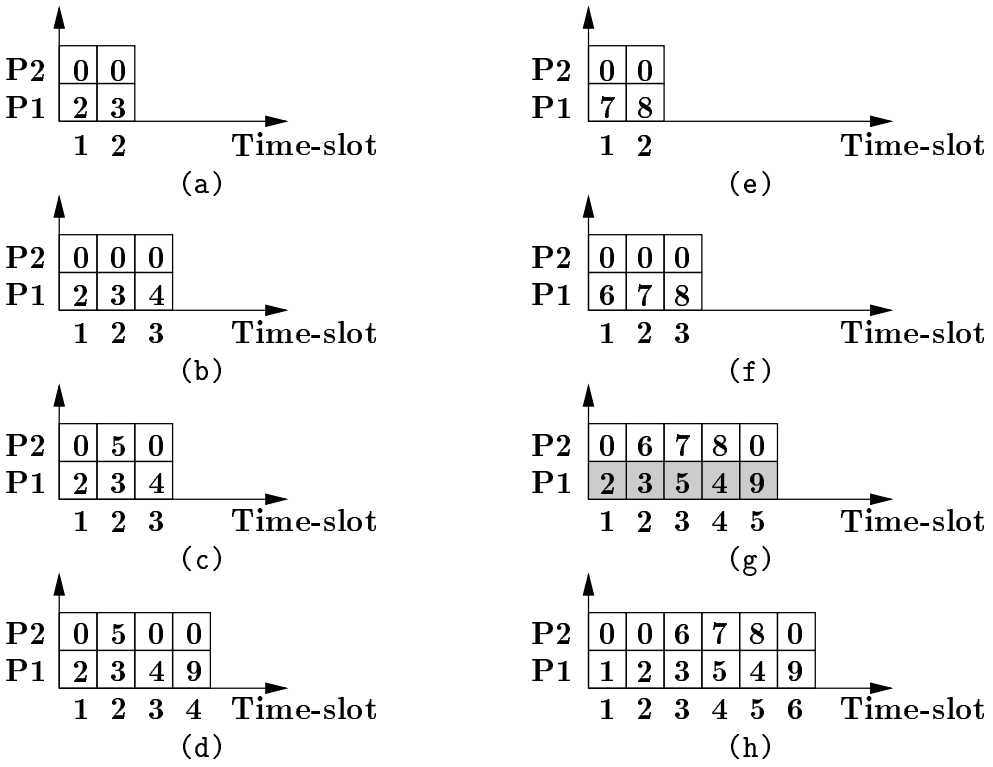

Figure 2: Intermediate nice schedules for the subgraphs of the task graph in Figure 1.

4 Concluding remarks

We have presented a quadratic time algorithm for the optimal schedule of SP1 graphs with UET-UCT on two processors. This result is a generalization of previous work on tree task graphs.

Note that our algorithm works also for graphs whose transitive reduction is an SP1 graph. In this case we have to find the transitive reduction of the given graph first, and then test if it is SP1, and finally apply the scheduling algorithm.

A remaining open question is the NP-hardness of the scheduling of general task graphs with UET-UCT on two processors. We conjecture however that this problem is polynomial when the task graph belongs to the class of general series-parallel graphs (without the restriction in the series composition).

References

[1] H. M. Abdel-Wahab, T. Kameda, "Scheduling to Minimize Maximum Cumulative Cost Subject to Series-Parallel Precedence Constraints", *Operations Research*, **26**, (1978) 141-158.

[2] P. Chretienne, C. Picouleau, "Scheduling with Communication Delays: A Survey", In *Scheduling Theory and Its Applications*, P. Chretienne et al. (Eds.), J. Wiley, 1995.

[3] L. Finta, Z. Liu, "Scheduling of Parallel Programs in Single-Bus Multiprocessor Systems", Rapport de Recherche INRIA, No. 2302, 1994, Submitted.

[4] R. L. Graham, E. L. Lawler, J. K. Lenstra, K. Rinnooy Kan, "Optimization and Approximation in Deterministic Scheduling: A Survey", *Ann. Disc. Math.*, **5** (1979), 287-326.

[5] F. Guinand, D. Trystram, "Optimal Scheduling of UECT Trees on Two Processors". Technical Report APACHE RR-93-03, IMAG, Grenoble, 1993.

[6] E. L. Lawler, "Sequencing Jobs to Minimize Total Weighted Completion Time Subject to Precedence Constraints", *Annals of Discrete Mathematics*, **2** (1978), 75-90.

[7] E. Lawler, "Scheduling Trees on Multiprocessors with Unit Communication Delays". In *Proc. Workshop on Models and Algorithms for Planning and Scheduling Problems*, Villa Vigoni, Lake Como, Italy, June 14-18, 1993.

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

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

[10] R. A. Sahner, K. S. Trivedi, "SPADE: A Tool for Performance and Reliability Evaluation", In *Proc. Modelling Techniques and Tools for Performance Analysis'85*, Ed. N. Abu El Ata, (1986) 147-163.

[11] K. Takamizawa, T. Nishizeki, N. Saito, "Linear-Time Computability of Combinatorial Problems on Series-Parallel Graphs", *JACM*, **18**, (1982) 623-641.

[12] J. Valdes, R. E. Tarjan, E. L. Lawler, "The Recognition of Series Parallel Digraphs", *SIAM J. of Computing*, **11**, (1982) 298-313.

[13] T. Varvarigou, V. P. Roychowdhury, T. Kailath. "Scheduling In and Out Forests in the Presence of Communication Delays". In *Proc. Intern. Parallel Processing Symposium*, Newport Beach, CA,(1993) 222-229.

[14] M. Veldhorst, "A linear Time Algorithm to Schedule Trees with Communication Delays Optimally on Two Machines". Technical Report COSOR 93-07, Dep. of Math. and Comp. Sci., Eindhoven Univ. of Technology, 1993.

[15] B. Veltman, "Multiprocessor Scheduling with Communication Delays", Ph.D. Thesis, CWI-Amsterdam, (1993).

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