ELSEVIER

Discrete Applied Mathematics 72 (1997) 5–24

DISCRETE APPLIED MATHEMATICS

# Some models for scheduling parallel programs with communication delays

Evripidis Bampis[a], Frédéric Guinand[b], Denis Trystram[b,*]

[a] *LaMI, Université d'Evry, Boulevard des Coquibus, 91025 Evry Cedex, France*
[b] *LMC-IMAG, 46 Avenue Félix Viallet, 38031 Grenoble Cedex, France*

Received 1 June 1994; revised 3 November 1995

**Abstract**

The aim of this paper is to present and analyze models for designing parallel programs. In the context of some extensions of the most popular execution models (precedence graphs, dataflow, PRAM), we describe scheduling techniques which take into account the communication delays. We illustrate all these models by two families of representative precedence graphs, namely, grids and complete trees.

*Keywords*: Parallel processing; Scheduling, Communications; DAGs, Grids; Complete trees

## 1. Introduction

The increasing development of parallel machines and their diversity make further investigations necessary in order to design and evaluate parallel algorithms. In the last ten years, several researchers attempted to model parallel machines. This effort coincides with an increasing interest in the study of the communications impact on the efficiency of parallel algorithms. Given the diversity of the actual parallel machines, several abstract models have been proposed taking into account the communications in rather different ways. Some of the proposed abstract models take into consideration nonuniform memory accesses, due to the coexistence of different types of memories, while others express the communication cost as a function of the distance between the communicating processors. In this paper, we present the most important models and we try to point out their common characteristics and their principal differences.

In the parallel processing context, a computational problem is often represented as a directed acyclic graph (DAG) [10] where the vertices represent the operations and the arcs the precedence constraints among the operations. Such a representation is a very useful tool since it provides a simple expression of potential parallelism. It has been used in several contexts, like the abstract interpretation of a program, of data flow

* Corresponding author. E-mail: Denis.Trystram@imag.fr.


*PII* S0166-218X(96)00034-1

computations, or of PRAM algorithms. Scheduling the tasks of a DAG constitutes one of the most important problems in parallel processing. Most scheduling problems are known to be NP-complete, even in the case where the communication cost is not taken into consideration [15].

Precedence task graph is the basic tool for static analysis of a program at compile time [10]. Within this computational model, the problem of efficiently parallelizing a sequential program or implementing a parallel algorithm can be reduced to the problem of determining a schedule with, if possible, the minimum execution time.

Data-flow is among the most important models in parallel computation. It is a language-based model of parallelism where the computations to be performed are specified by a DAG in which each node corresponds to an operation and each arc to a data transmission.

PRAM is the most popular abstraction of parallel machines. It models a shared-memory parallel machine where the access time from any processor to any memory location is constant. Unfortunately, PRAM is not representative of most actual parallel machines. Although the practical interest of the PRAM model has to be demonstrated (maybe with the development of optics) its theoretical interest remains fundamental and thus, many researchers still design new parallel algorithms within the PRAM model. These parallel algorithms can be emulated by most actual parallel machines by the appropriate use of message routing [22]. However, in the time complexities of the PRAM algorithms an additional overhead must be added due to the communication delay. This delay depends on the data routing through the interconnection network of the parallel machine.

In an effort to capture the influence of the communication delays, many extensions of the above abstract models have been proposed. We present below some of the most important extensions, and we point out the links between them.

The first model (presented in Section 2) was developed by Anderson et al. for parallel shared-memory machines [2]. Section 3 is devoted to an architecture-independent model which allows recomputation of tasks. It was introduced by Papadimitriou and Ullman in 1987 [24]. At the same time, Rayward-Smith considered in [27] the problem of scheduling graphs with unit execution time (UET) tasks and unit communication delays. This model allows to overlap the communications by the computations. One year later, Papadimitriou and Yannakakis [25] proposed a generalized description of this model considering both the possibility to use recomputation in order to reduce the communication cost, and the overlap of the communications by the computations. This model is presented in Section 4. Following a short description of all these models, we then propose the same study on extensions of the PRAM model. Thus, in Section 5, we first present the Aggarwal–Chandra–Snir's model [3], as well as two more extensions, respectively, due to Cosnard and Ferreira [12], and Bampis et al. [8].

As an illustration of all these models, some results and examples concerning the scheduling of complete trees and grids are given at the end of each section. For the sake of completeness, let us define formally these two families of DAGs.

We define a *two-dimensional grid* precedence graph of size $(n_1, n_2)$, 2D-grid $(n_1, n_2)$ in short, as the set of pairs $(i,j)$, where $0 \leqslant i \leqslant n_1 - 1$, $0 \leqslant j \leqslant n_2 - 1$. Each vertex of the grid is a unit execution time task, denoted as UET, and task $(i,j)$ precedes the tasks $(i+1,j)$ and $(i,j+1)$, if these tasks exist. In the following, the term 2D-grid will be used to represent the square grid of size $(n,n)$. This definition can be extended to multidimensional $d$D-grids (with $d \geqslant 3$).

A *complete k-ary tree* of depth $h$, denoted $\mathcal{T}_h$, is a graph whose vertices are all the words of length not greater than $h$ constructed on an alphabet of $k$ letters $\{0,1,\ldots, k-1\}$. Each edge connects a vertex which is represented as a work $x$ of length $i$ with the vertices $x\alpha$, $\alpha \in \{0,1,\ldots,k-1\}$, of length $i+1$, $0 \leqslant i < h$. The vertices which are represented by a word of length $i, 0 \leqslant i \leqslant h$, form the level $i$ of the tree. The complete $k$-ary tree $\mathcal{T}_h$ has $(k^{h+1}-1)/(k-1)$ vertices. As in the case of grids, each vertex is a UET task. Whenever the orientation of the edges of the tree goes from vertex $x$ to vertex $x\alpha$, we talk about *out-trees*, while when the orientation goes from $x\alpha$ to $x$, we talk about *in-trees*.

## 2. Anderson–Beame–Ruzzo (ABR) model

### 2.1. Presentation

This model has been introduced by Anderson et al. in the context of small parallel shared-memory machines [2]. In this case, two sources of overhead are relevant: the processors' idle time and the time for performing synchronization or scheduling.

Anderson et al. consider that the synchronization overhead is proportional to the number of tasks of the DAG. This assumption is also valid in the case of shared-memory machines with a large number of processors where communication problems occur [4, 6]. In fact, in a shared-memory multiple instructions multiple data (MIMD) architecture, the execution of a task consists of three steps: the data loading from the shared memory, the computation of the task and the output of the results to the memory. If we assume that a processor cannot exchange information with the shared memory during the execution of a task, and that each memory access takes a constant time (whatever is the volume of transfered data), then the communication cost is proportional to the number of tasks. This assumption is realistic in the case where the loading of variables and the output of results is pipelined.

For a particular computational problem represented as a DAG, the objective of an efficient schedule is to minimize the *parallel execution time*, also known as *makespan*, defined as $(T_{\text{seq}} + T_{\text{idle}} + T_{\text{com}})/m$, where

- $T_{\text{seq}}$ is the sequential time for the execution of the DAG. In the case of DAGs with UET tasks, $T_{\text{seq}}$ is equal to the number of nodes of the graph.
- $T_{\text{idle}}$ is the total idle time of the processors during the execution of the DAG,
- $T_{\text{com}}$ denotes the communication cost and,
- $m$ the number of processors.

Since $T_{seq}$ and $m$ are considered as given, the minimization of the makespan is obtained by minimizing $(T_{idle} + T_{com})$. This parameter is called *overhead.*

In order to measure $T_{com}$, the notions of *job* and *schedule graph* were introduced in [2]. A *job* $J$ was defined as a set of elementary tasks executed by the same processor. The execution of $J$ consists in the three steps previously described. It is then obvious that reducing the communication needs a "nice" gathering of the tasks into a small number of jobs. If the gathering is not made carefully, it may lead to the increase of the other source of overhead, i.e. the processors' idle time. For instance, let us consider the partition that minimizes the communication cost: the whole DAG is considered as a single job. In this case, the communication cost is minimized, but there will be all but one processor idle during the sequential execution of the unique job. At the other extreme, if each elementary task is considered as a job, then the processors' idle time is reduced since in the UET task graph there is the maximum potential parallelism, but at the same time the communication/synchronization cost is maximized since it becomes equal to the number of tasks of the graph, i.e. to the sequential time.

Let us notice that a job cannot exchange information with the memory during its computation. By gathering the tasks into jobs, the precedence constraints among tasks are transformed into precedence constraints among jobs. For instance, if task $v$ belongs to job $J_1$ and $u$ to job $J_2$ and there is an arc from $v$ to $u$ in the initial DAG, then an arc must be added from $J_1$ to $J_2$. Hence, a new graph is obtained: the schedule graph whose vertices are jobs and whose arcs are the precedence constraints among the jobs. It is assumed that the gathering is made in such a way that the resulting schedule graph contains no circuits, i.e. it remains a DAG. Moreover, scheduling jobs instead of tasks reduces the potential parallelism by forcing an increased number of tasks to be executed sequentially, but, since there are no communications inside the same job, the reduction of the communications is hoped. Given that each job needs a constant time to communicate with the memory, the communication cost is proportional to the number of jobs, and the overhead is simply the sum of the idle time of all the processors and of the number of jobs. Then, the objective is to find strategies for gathering the UET tasks into jobs, and for the assignment of these jobs into the processors in order to minimize the overhead, i.e. to find a trade-off between $T_{idle}$ and $T_{com}$.

### 2.2. Example

In Fig. 1, we consider a simple example of a DAG with seven UET tasks and we study two different gatherings of the tasks into jobs.

The execution time of each job, in the corresponding schedule graph, includes the communication cost, equal here to one. The sequential time in both cases is $(T_{seq} =)$ 7. For the first gathering, if we consider two processors, we have $(T_{com} =)$ 3 and $(T_{idle} =)$ 4, thus an overhead equal to 7. For the second, we obtain $(T_{com} =)$ 4 and $(T_{idle} =)$ 1, thus an overhead equal to 5.

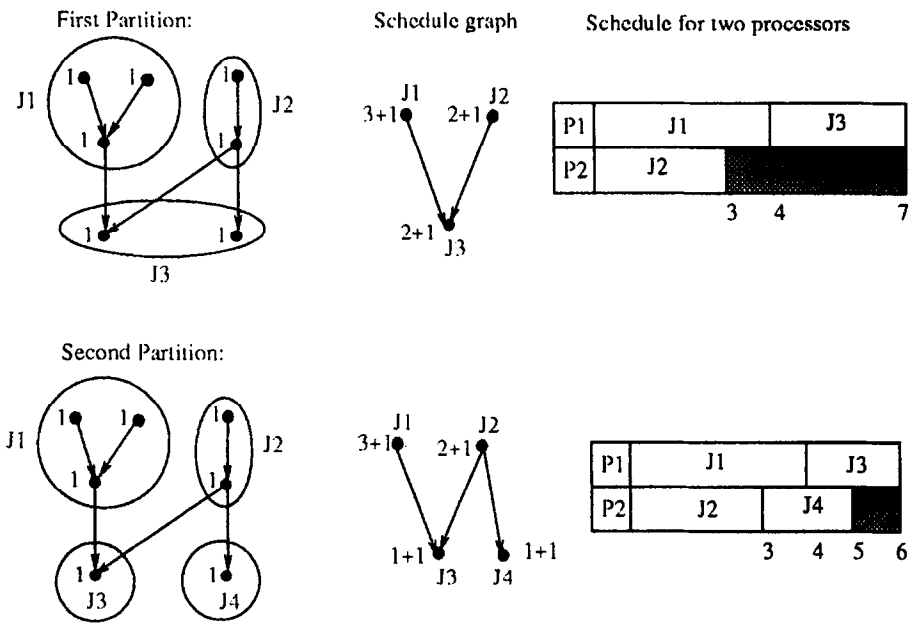



Fig. 1. Two different gatherings of a precedence graph with unit execution time tasks.

## 2.3. Application to trees and grids

### 2.3.1. Trees

The first asymptotically optimal algorithm for scheduling a complete binary tree on a fixed number of processors was proposed by Bampis and König [4]. For a given number of processors, say $m$, and for a complete binary tree of depth greater than $\log m$, the overhead is in $O(m \log m)$. This value of the overhead reaches the lower bound for the overhead of any schedule of a complete binary tree of depth greater than $\log m$. An algorithm that provides an optimal schedule for complete $k$-ary trees on a fixed number of processors was presented in [16]. This algorithm uses the same basis as the one in [4], but the resulting schedule not only in asymptotically optimal, but also reaches the exact optimal value of the minimum makespan.

Let us sketch the principle of the algorithm presented in [4]. Given a number $m$ of processors and a $k$-ary complete out-tree, the algorithm consists of two stages. The idea is to minimize the idle time in the first stage and the number of jobs in the second one. Thus, the first one deals with the first part of the complete tree where each level contains less than $m$ tasks. Each UET task is considered as a job, and the execution is made level by level. The second stage provides a schedule for a forest of complete trees on the $m$ processors. Let $z$ be the number of tasks of the first level of the tree that contains more than $m$ tasks. We assign $z/m$ complete trees to the $m$ processors (if $z$ is a multiple of $m$), or $\lfloor z/m \rfloor$ complete trees to $m_1$ processors and $\lceil z/m \rceil$ partial subtrees to $m_2$ processors, with $m_1 + m_2 = m$, and where the number of tasks assigned to each

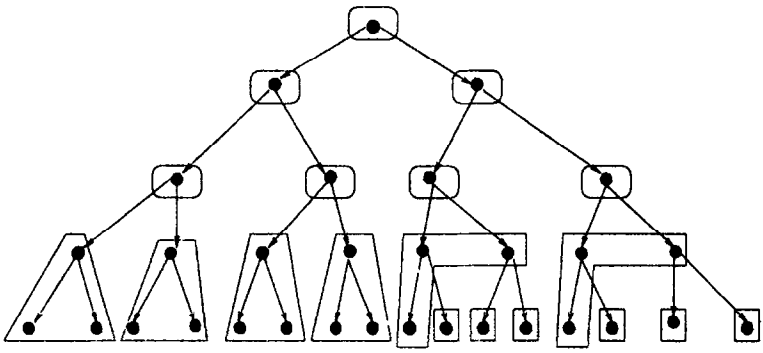


Fig. 2. Partition of a complete binary tree of depth $h = 5$ for 6 processors. It is easy to verify that $T_{com} = 19$ and $T_{idle} = 22$.

processor is exactly the same. The allocation is done in such a way that the remaining tasks, if any, belong to the last level of the tree, in order to be independent of each other (this is always possible because of the tree structure). Let $q$ be the number of the remaining tasks, if any. These $q$ tasks are partitioned into $m$ independent parts of equal size (if $q$ is a multiple of $m$), or of sizes equal to $\lfloor q/m \rfloor$ and $\lceil q/m \rceil$ (if $q$ is not a multiple of $m$). An illustration of the algorithm is given in the example of Fig. 2. It is easy to verify that the overhead of this schedule is in only $O(m \log m)$. The detailed proof of the asymptotic optimality of the algorithm is given in [4].

### 2.3.2. *Grids*

Let us now consider a 2D-grid. If we consider $m$ ($<n$) processors and a partition where each UET task of the grid is considered as a job, then the idle time is $T_{idle} \geqslant 2\sum_{i=1}^{m}(m-i) = m^2 - m$, and the communication cost is equal to ($T_{com} =$) $n^2$. On the other hand, if we partition the grid into boxes of size $w$ by $w$ (assume for simplicity that $n$ is a multiple of $w$), then, if $m < n/w$, we obtain $T_{idle} \geqslant (m^2 - m)w^2$ and $T_{com} = n^2/w^2$. We can see that the increase of the jobs' size leads to the decrease of the communication cost, but on the other hand there is an augmentation of the idle time. For this kind of partition, the trade-off is obtained for $w = \sqrt{(n/m)}$ leading to an overhead of $O(nm)$. However, as it has been shown in [2], this result can be improved. In fact, using a recursive partition, Anderson Beame and Ruzzo [2] succeeded to reduce the overhead to only $O(\log \log n)$ where $n$ is the size of the grid, and proved that $\Omega(\log \log n)$ is a lower bound for any scheduling of the 2D-grid using a fixed number of processors $m$. Bampis et al. [7] have extended these results in the case of 3D-grids. In addition, they have studied the problem of minimizing the overhead for 3D-grids when the number of processors is a function of the grid size ($m = \alpha n$, $0 < \alpha \leqslant n$) [9]. In this case, the best known strategy considers adaptive granularity jobs leading to an overhead in $O(n^{5/3})$.

Table 1 summarizes the main results concerning scheduling of grids and complete trees for the current model.

Table 1
Execution of grids and trees within the ABR model

| Algorithm-DAG | Lower bound | Upper bound | $T_{seq}$ | Number of processors |
|---|---|---|---|---|
| 2D-grid [2] | $\Omega(\log\log n)$ | $O(\log\log n)$ | $n^2$ | $m$ fixed |
| 3D-grid [7] | $\Omega(\log\log n)$ | $O(\log\log n)$ | $n^3$ | $m = 2, 3$ |
| 3D-grid [9] | $\Omega(n^{3/2})$ | $O(n^{5/3})$ | $n^3$ | $m = \alpha n,\ 0 < \alpha \leqslant 1$ |
| Compl. bin. Trees [4] | $\Omega(m \log m)$ | $O(m \log m)$ | $2^{h+1} - 1$ | $m$ arbitrary |

## 3. Papadimitriou–Ullman (PU) model

### 3.1. Presentation of the model

Papadimitriou and Ullman [24] introduce the nonuniform memory access assumption. They distinguish between two different types of memory access: an access of each processor to its local memory with zero communication cost, and a distant access with a communication cost equal to one time unit. The authors try to develop an architecture-independent methodology, and they propose two different parameters for estimating the performance of an algorithm. These parameters are the execution time (without taking into account the communications) and the communications. In this model the program is represented as a DAG with UET tasks. During the execution of the DAG, each task can be computed by one or more processors, i.e. a model where recomputation (also known as duplication) is allowed.

Given an allocation of the tasks into the processors, a distant communication is represented as an arc, the extremity nodes of which are executed by two different processors. More formally, the authors introduce a relation between processor–task pairs, denoted by $D$. Assume that the task $u$ is executed on processor $P$ and that the task $v$ is a predecessor of $u$. The processor–tasks pair $(P,u)$ depends on the pair $(P',v)$ if $P'$ has computed $v$ before $P$ computes $u$. If several $P'$ exist, then only one is taken into account for each processor–task pair $(P,u)$ and task $v$. Three parameters are considered as relevant for the performance evaluation of an algorithm:

- the execution time without taking into account the communications, denoted as $T_{\text{comput}}$,
- the total number of communications (total message traffic), denoted as $T_{\text{traffic}}$,
- the maximum number of communications in any oriented path of the DAG (total elapsed time due to communications), denoted as $T_{\text{delay}}$.

Using the notion of communication arc and given a processor allocation, $T_{\text{traffic}}$ corresponds to the total number of communication arcs in the DAG, and $T_{\text{delay}}$ represents the maximum number of communication arcs in any path of the DAG. Papadimitriou and Ullman consider that a relevant criterion expressing the trade-off of the communi-

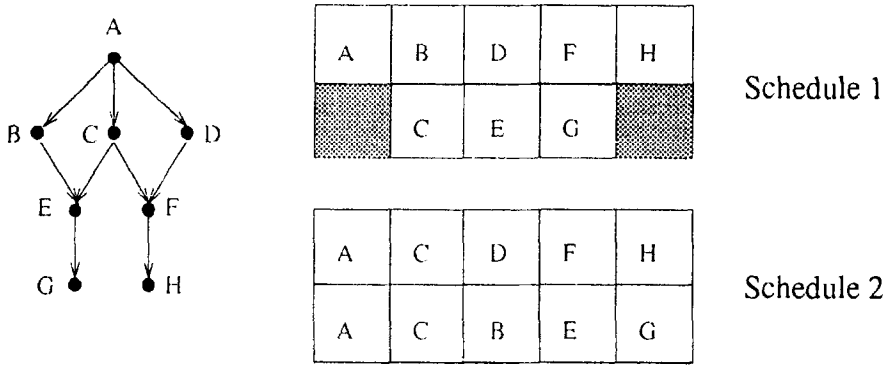



Fig. 3. An example for the PU model without duplication (Schedule 1) and with duplication (Schedule 2).

cations and the computations is the product of the communication cost ($T_{\text{traffic}}$ or $T_{\text{delay}}$) by the computation time $T_{\text{comput}}$.

Given these three parameters, several other criteria can be examined in order to evaluate the performance of an algorithm. If the primary importance aim is to avoid the congestion of the interconnection network or of the bus, and at the same time a fast execution is required, then the minimization of $(T_{\text{comput}}+T_{\text{traffic}})$ is a pertinent goal. If we are only interested in the minimization of the makespan (i.e. the time when the last task of the DAG will finish its execution), then $(T_{\text{comput}} + T_{\text{delay}})$ is the pertinent criterion.

Thus, the efficient use of the target architecture depends on the resolution of optimization problems such as

- minimizing the makespan. As the overlapping of the communications by the computations is not allowed, the makespan is equal to $(T_{\text{comput}} + T_{\text{delay}})$, where $T_{\text{comput}}$ is here the longest path of the DAG, i.e. the time needed for the execution of the DAG when an unbounded number of processors is considered. Thus, the minimization of the makespan is obtained by minimizing $T_{\text{delay}}$.
- minimizing $T_{\text{traffic}}$ when an unbounded number of processors is considered ($T_{\text{comput}}$ is here also equal to the longest path of the DAG).
- minimizing both $T_{\text{traffic}}$ and $T_{\text{delay}}$, under the same assumption.

The same kind of problems when the number of processors is either fixed or an entry of the problem constitutes a very interesting area of future research. Recall here that the subproblem of minimizing $T_{\text{comput}}$ when the number of processors is fixed (greater than two) is one of the most important open problems in computer science.

### 3.2. Example

We consider a simple example of a DAG with eight UET tasks (see Fig. 3). We want to schedule it in five time units.

In Fig. 3, the value of $T_{\text{traffic}}$ in Schedule 1 is equal to the total number of communication arcs, here 3 $\{(A,C),(B,E),(C,F)\}$. $T_{\text{delay}}$ is equal to the maximum number

of communication arcs in any path of the DAG. The path containing the maximum number of communication arcs is the path (A, C, F, H) where two communication arcs exist, namely (A, C) and (C, F). So, $T_{\text{traffic}} = 3$ and $T_{\text{delay}} = 2$.

In Schedule 2 by using recomputation, we obtain $T_{\text{traffic}}$ and $T_{\text{delay}}$ equal to zero.

### 3.3. Some results

#### 3.3.1. Trees

If we consider an unbounded number of processors, and we adopt as objective the minimization of either $(T_{\text{comput}} + T_{\text{traffic}})$ or $(T_{\text{comput}} + T_{\text{delay}})$, then an optimal solution for a complete $k$-ary out-tree of depth $h$ is given by using duplication and simply assigning one processor to each duplicated path of the out-tree. This implies that the root is duplicated $k^h$ times. This algorithm needs as many processors as the leaves of the tree, and guarantees that $T_{\text{traffic}} = T_{\text{delay}} = 0$, and that $T_{\text{comput}}$ is equal to the length of the longest path of the tree (without taking into account the communications).

Under the same assumptions, the problem of scheduling a complete $k$-ary in-tree is more difficult, since duplication is useless. Let us consider a $k$-ary in-tree of depth $h$ with $n = (k^{h+1}-1)/(k-1)$ vertices. If only one processor is considered, then $T_{\text{comput}} = n$ and $T_{\text{traffic}} = T_{\text{delay}} = 0$, so, $T_{\text{comput}} + T_{\text{traffic}} = T_{\text{comput}} + T_{\text{delay}} = n$. On the other hand, if we consider as many processors as the leaves of the tree (i.e, $k^h$ processors), then a level-by-level execution gives $T_{\text{comput}} = h + 1$, $T_{\text{traffic}} = k^h - 1$ (there is only one processor executing the root), and $T_{\text{delay}} = h$ (at each level, every processor executes the root of a fork, and thus only one of its arcs is not a communication arc). Finally, $(T_{\text{comput}} + T_{\text{traffic}}) = h + 1 + k^h - 1 = h + k^h$, and $(T_{\text{comput}} + T_{\text{delay}}) = h + 1 + h = 2h + 1$.

In [24], the authors give the following lower bound for the execution of a complete binary tree with $n$ nodes: $T_{\text{comput}} T_{\text{traffic}} = \Omega(n)$. Afrati et al. [1] considered a restricted version of the PU model where the recomputation of tasks is not allowed. They studied the following problem: Given a DAG, a number of processors $m$ (fixed, part of the problem, or not limited), and two positive integers $t$ and $c$, is it possible to schedule the DAG on $m$ processors in time $t$ and total message traffic $c$?. They proved that in the case of trees the problem is strongly NP-complete (when the number of processors is not limited or is part of the input), and it remains so, in the case of general DAGs and only 2 processors.

#### 3.3.2. Grids

In the case of a 2D-grid task graph, one solution is to schedule the tasks on $m \leqslant n$ processors by partitioning the grid into stripes [24]. More precisely, the grid is divided into $m$ stripes of height $n$ and width $n/m$, i.e. each processor executes $n^2/m$ tasks (assume for simplicity that $m$ divides exactly $n$). This partition is called the *stripes* method. At the beginning, only one processor can work because of the precedence constraints. Then, at time $ni/m$, $0 \leqslant i \leqslant m-1$, the $(i+1)$th processor can start working and so after $n$ units of time all the processors will be active. The computation time without taking into consideration the communication delays is $T_{\text{comput}} = (n^2/m) +$

Table 2
Lower bounds for the execution of 2D-grids within the PU model

| $(T_{\text{traffic}} + n)T_{\text{comput}}$ | $(T_{\text{delay}} + 1)T_{\text{comput}}$ |
|---|---|
| $\Omega(n^3)$ | $\Omega(n^2)$ |

$(m-1)(n/m)$, the total communication traffic is $T_{\text{traffic}} = (m-1)n$, and the total communication delay is $T_{\text{delay}} = (m-1)$.

Another strategy, proposed in [24], is to partition the grid into boxes of size $n/\sqrt{m}$ by $n/\sqrt{m}$, i.e. each processor executes $(n/\sqrt{m})(n/\sqrt{m})$ tasks (for simplicity assume that $n$ is a multiple of $\sqrt{m}$). In this case, $T_{\text{comput}} = O(n^2/\sqrt{m})$ since only one processor can be active in a column at any time, the total communication traffic is $T_{\text{traffic}} = 2n(\sqrt{m}-1)$, and the total communication delay is $T_{\text{delay}} = 2(\sqrt{m}-1)$.

The main results of [24] are given in Table 2.

## 4. Papadimitriou–Yannakakis (PY) model

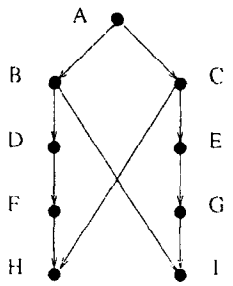
### 4.1. Presentation of the model

Papadimitriou and Yannakakis consider a data-flow model where the computational problem is represented by a DAG [25], and try to identify a unique parameter in order to model a broad class of parallel machines. The basis of their analysis is the communication delay between the time one task is computed on a processor and the time the produced value can be used by another processor. This communication delay, denoted by $\tau$, is measured in elementary steps of the processors and it is considered as the fundamental parameter of the architecture. In fact, it represents the ratio of the communication cost for the transfer of an elementary data to the execution time for an elementary instruction. In the case of shared-memory parallel machines, $\tau$ is constant, while when distributed-memory machines are considered, $\tau$ depends on the number of processors and on the topology of the underlying interconnection network. For instance, for a parallel machine with $m$ processors and for an interconnection network which is a ring, $\tau$ is in $O(m)$. When the interconnection network is a grid (resp. a hypercube), $\tau$ is in $O(\sqrt{m})$ (resp. in $O(\log m)$).

If the value of $\tau$ is given and under the assumption of a nonlimited number of processors, then an efficient parallelization of the computation needs the minimization of the *makespan* (maximum finishing time), denoted by *Cmax*, in the following scheduling problem:

Schedule a DAG, with UET tasks, so that if task $i$ starts its execution at time $t$ on $P$ and $j$ is a successor of $i$, then either $j$ starts its execution at time $t+1$ on processor $P$, or $j$ starts at time $t+1+\tau$ on some other processor, say $P'$.

Notice that in contradiction with the ABR and PU models, a communication appears only, if it causes an extension of the makespan. Indeed, in the previous




| | | | | |
|---|---|---|---|---|
| A | C | E | G | I |
| A | B | D | F | H |

Fig. 4. We assume that $\tau = 2$. The duplication of task $A$ allows to avoid one communication (either from $A$ to $B$ or from $A$ to $C$) and the overlap allows the communication from $B$ to $I$ (resp. from $C$ to $H$), and the computation of $D$ and $F$ (resp. of $E$ and $G$) to be done simultaneously.

example, if we decide to execute $j$ on processor $P'$ and if $P'$ is active during the time interval from $t+1$ to $t+1+\tau$, then the communication from $P$ to $P'$ "disappears", since it does not increase the makespan of the schedule. Thus, in the PY model, we have the possibility to *overlap* the communications by the computations. This is a quite reasonable assumption since in practice, MIMD machines are asynchronous.

Many variations of this model exist in the literature. For instance, Rayward–Smith, in [27], studied the case where $\tau$ is equal to the time unit, the recomputation of tasks is not allowed and the number of processors is a parameter of the problem. This model is also known as the UET-UCT or RS model. Colin and Chrétienne [11] considered another variant of the above model, known as the SCT (small communication times) model, where the largest communication delay is less than or equal to the smallest execution time of the tasks, and where recomputation is allowed. They proved that in this case minimizing the makespan is an "easy" problem by providing a polynomial-time algorithm.

### 4.2. Example

In the example of Fig. 4, we consider the PY model. In order to obtain the optimal schedule both overlapping and duplication are necessary.

In the example of Fig. 5, we consider the RS model. It is not possible to compute the whole graph with an overall computation time less than 5. This time is reachable with two processors. Notice that the communication from $B$ to $E$ is overlapped by the computation of task $C$ and the communication from $D$ to $G$ is overlapped by the computation of task $F$.

### 4.3. Some results

In [25], an approximation algorithm is proposed for the PY model, which gives the optimum makespan within a factor of two when recomputation is allowed.

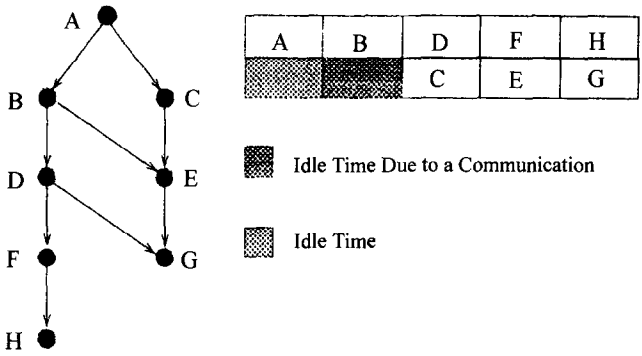



Fig. 5. Optimal schedule using overlapping within the RS model.

Table 3
Results for the execution of trees within the PY model

| Problem formulation | Lower bound | Reference | Complexity |
|---|---|---|---|
| $P_\infty \mid compl.\ bin.\ intrees, p_i = 1, \tau_{ij} \mid Cmax$ | | [18] | NP-complete |
| $P_\infty \mid compl.\ k\text{-}ary\ intrees, p_i = 1, \tau(n) \mid Cmax$ | | [18] | $O(n^2 \log(n))$ |
| $P_\infty \mid outtrees, p_i, \tau_{ij} \mid Cmax$ | | [11] | $O(n)$ |
| $P_\infty \mid compl.\ bin.\ intrees, p_i = 1, \tau(n) \mid Cmax$ | $Cmax_{\text{opt}} \geqslant 2\tau \frac{\log(n)}{\log(\tau)}$ | [19] | |
| $P_\infty \mid compl.\ bin.\ intrees, p_i = 1, \tau > 1 \mid Cmax$ | | [14] | $O(n)$ |

Table 4
Results for the execution of trees within the RS model

| Problem formulation | Reference | Complexity | Remark |
|---|---|---|---|
| $P \mid trees, p_i = 1, \tau = 1 \mid Cmax$ | [26, 31] | NP-complete | |
| $P_m \mid trees, p_i = 1, \tau = 1 \mid Cmax$ | [21] | $O(n)$ | Fixed $m$, $Cmax \leqslant Cmax_{\text{opt}} + (m-2)$ |
| $P_m \mid trees, p_i = 1, \tau = 1 \mid Cmax$ | [28] | $O(n^{2(m-1)})$ | Fixed $m$ |
| $P_2 \mid trees, p_i = 1, \tau = 1 \mid Cmax$ | [17, 29, 30] | $O(n)$ | |

### 4.3.1. Trees

The application of the approximation algorithm of [25] in the case of a complete binary in-tree with $n$ tasks gives a partition of the tree into $O(\log n / \log \tau)$ layers of depth $O(\log \tau)$ or in other words, into $O(n/\tau)$ subtrees. All the subtrees of the same level are computed in parallel and the obtained makespan is in $O(\tau \log n / \log \tau)$ using $O(n/\tau)$ processors.

Jacoby and Reischuk [16] proposed an optimal algorithm for scheduling complete $k$-ary trees within the PY model (the communication cost $\tau$ may depend on the DAG, i.e. may increase with the number of the tasks of the DAG). In Table 3, we give the known results for the execution of in-trees within the PY model. Recently, many results have been established within the RS model; we present them in Table 4 (under the assumption that the number of processors is fixed). In both tables, we use the notation introduced in [32].

*4.3.2. Grids*

Using the same approximation algorithm of [25], the makespan required for the computation of a 2D-grid is $O(2n\sqrt{\tau})$ with $O(n - \sqrt{\tau})$ processors.

Norman et al. in [23], considered both the PU and the PY models. They compare within these two models three different scheduling of 2D-grids. The two first methods are the methods *stripes* and *boxes* proposed in [24] while the third one, considers a partition of the grid into *lines*: all the tasks of column $j$ are executed by processor $(j \bmod m)$. Although lines has a larger value of $T_{\text{traffic}}$ and $T_{\text{delay}}$ than stripes and boxes within the PU model, it gives the smallest value of makespan.

Bampis et al., in [5], studied the more general case of $d$D-grids. They proved that in the case where the number of processors is not bounded, a lower bound of the makespan, for any scheduling algorithm, is equal to $(2d - 1)(n - 1) + 1$ and that this lower bound can be reached by the schedule by *lines* for $m = n$. Furthermore, they show that *lines* schedule is the only schedule able to execute a $d$D-grid in the optimal time and they compute the exact value of the minimum number of processors required to execute a $d$D-grid optimally. For instance, for a 2D-grid this number is $\lceil n/2 \rceil$.

## 5. Aggarwal–Chandra–Snir (ACS) model

### *5.1. Presentation of the model*

In the PRAM model the communication costs are "hidden" in the computation steps (every computation step includes three stages: the data loading from the shared memory, the execution and the output of the results into the memory). Aggarwal et al. [3] have proposed a new approach for estimating the impact of communications within the PRAM model. Their model is known as the Local-memory PRAM (LPRAM) or ACS model and models a concurrent-read, exclusive-write (CREW) PRAM, where each processor has an unlimited local memory. As in the case of the PU and PY models, the authors consider the nonuniform access memory assumption: an access to the local memory is considered with zero communication cost while a memory access in the shared memory costs one time unit.

The execution of every program is preceded by the data loading from the shared memory and succeeded by the output of the results. The processors can execute different programs on different data (MIMD) but they work synchronously. An execution can be described as a set of unit communication and unit computation phases. During one communication phase, the processors read (and/or write) an elementary data to (from) the shared memory, while during a computation step the processors execute an elementary instruction. The computational problem is represented by a DAG with UET tasks. A task of the DAG can be executed on processor $P$, only when all the values corresponding to the incoming arcs of the task are contained in the local memory of $P$.

Notice that the ACS model is very close to the PU model. The main difference is that each processor can have at most one communication request outstanding at any

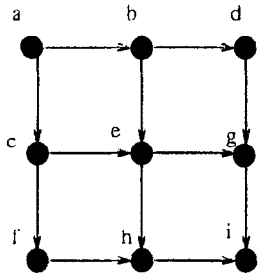



Fig. 6. In order to execute this 2D-grid on three processors, the following computation and communication steps are required:
*Communication Step 1*: $P_1$ reads $a$.
*Computation Step 1*: $P_1$ computes $a$.
*Communication Step 2*: $P_1$ writes $a$ and $P_2$ reads $a$.
*Computation Step 2*: $P_1$ and $P_2$ compute $b$ and $c$, respectively.
*Communication Step 3*: $P_1$ and $P_2$ write $b$ and $c$ and $P_1$ and $P_3$ read $a$ and $b$, respectively,
*Computation Step 3*: $P_1$, $P_2$ and $P_3$ compute $e$, $f$ and $d$, respectively.
*Communication Step 4*: $P_1$, $P_2$ and $P_3$ write $e$, $f$ and $d$, respectively, and $P_1$, $P_2$ read $d$ and $e$, respectively,
*Computation Step 4*: $P_1$ and $P_2$ compute $g$ and $h$, respectively.
*Communication Step 5*: $P_1$ and $P_2$ write $g$ and $h$, respectively, and $P_2$ reads $g$.
*Computation Step 5*: $P_1$ computes $i$.
*Communication Step 6*: $P_1$ writes $i$.

time. This restriction is very similar to the *1-port* assumption [13] in the context of distributed memory machines. Another difference is that recomputation is not allowed in the ACS model and that the processors work in a synchronous way.

In order to measure the performance of an algorithm, the criterion of the *total execution time* is used (defined as the sum of the number of computation and of communication phases). An example illustrating this model is depicted in Fig. 6.

### 5.2. Example

Let us consider the fork and join DAGs depicted in Fig. 7. We can see that for the fork, the possibility of recomputation offered by the PU model allows to eliminate all the communications. For the join and within the ACS model, as only one communication is outstanding during a communication step, three communication steps are necessary for the first processor in Schedule 3 in order to get all the necessary data for the computation of e. In the case of the join DAG duplication is useless. In fact in Schedule 4, we have $T_{\text{delay}} = 4$ but $T_{\text{traffic}} = 4$, i.e. equal to the number of communication steps in Schedule 3 (excepting the initial load and the final write from/to the shared memory).

### 5.3. Application to trees and grids

In the case of complete binary in-trees or 2D-grids, each task has at most one or two incoming arcs. Thus the total execution time in the ACS model, is equal to $T_{\text{comput}} + T_{\text{delay}} (= T_{\text{comput}} + T_{\text{traffic}})$ in the PU model. In both cases recomputation is

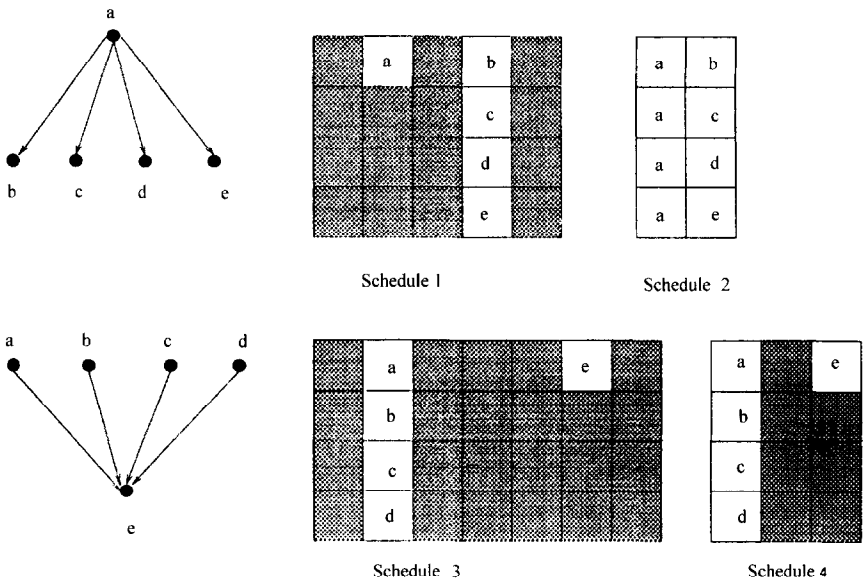



Fig. 7. Execution of a fork and a join graph within the ACS model (Schedules 1 and 3) and within the PU model (Schedules 2 and 4), on four processors.

useless, and the total number of communication steps (except the first read of data and the last write of results from/in the shared memory) of the ACS model is equal to $T_{\text{delay}}(= T_{\text{traffic}})$ of the PU model.

### 5.3.1. Trees

In [3], Aggarwal et al. have studied the computation of binary trees. For arbitrary trees with $n$ leaves and depth $h$, they proved that any algorithm using $m$ processors requires $\Theta((n/m)+h)$ computation steps and that the number of communication steps is at least

$$\Omega((n/m)+\log n+\sqrt{h}) \text{ and at most } O((n/m)+\min(\sqrt{n},h)).$$

They also provide an $O(n \log n)$ algorithm based on a partition of the original tree in subtrees and a labeling of the nodes of the tree (the label corresponds to the earliest communication step at which the value could be written into the shared memory). This algorithm computes a schedule for any binary tree $T_b$ that achieves a communication delay no more than $2D_{\text{opt}}(T_b)$ (where $D_{\text{opt}}(T_b)$ denotes the minimum communication delay for a given binary tree $T_b$). This result has to be compared with the algorithm of [13] which allows the execution of an arbitrary DAG within a factor of two from [25] the optimal makespan.

### 5.3.2. Grids

Aggarwal et al. [3] have studied also, the computation of 2D-grids and have extended the results of Papadimitrion and Ullman for 2D-grids by showing that $T_{\text{delay}}\, T_{\text{comput}} =$

$\Theta(n^2)$ can be achieved for essentially two values: $T_{\text{delay}} = \Theta(1)$ and $T_{\text{comput}} = \Theta(n^2)$ and $T_{\text{delay}} = \Theta(n)$ and $T_{\text{comput}} = \Theta(n)$.

### 5.4. Extensions

Cosnard and Ferreira [12] have proposed another extension of the PRAM model, namely the XRAM model, which takes into account the topology of the interconnection network. Their approach is based on the remark that a PRAM can be viewed also as an idealized distributed-memory model where the processors are connected by a complete interconnection network (i.e. a network where all pairs of processors are connected by a direct link). XRAM extends this model in the case of an arbitrary interconnection network X. Processor $P_i$ can only access its own memory locations or the memory locations of its neighbors. In one unit of time, each processor reads an elementary data, executes an elementary task and writes the result. If a processor $P_i$ needs a data from a processor $P_j$ which is not a neighbor, a routing must be used to transfer this data through the interconnection network to $P_i$.

Thus, XRAM offers a very useful tool for the implementation of PRAM algorithms in distributed memory architectures. However, in the time complexities of these PRAM algorithms, we must incorporate an additional overhead due to the data routing.

Bampis et al. [8] proposed an approach that takes into account the algorithmic structure by considering the notions of task and task precedence graph issued from the classical scheduling theory [10]. The goal of this model is to replace the routing in the XRAM (which is mostly the most expensive part of the implementation of an algorithm) by local communications between dependent instructions. We have seen above that an efficient implementation of a PRAM algorithm in a distributed-memory computer within the XRAM model might necessitate an additional overhead due to the data routing. The Bampis–König–Trystram (BKT) model considers an execution scheme where in one unit of time each processor executes exactly the same operations as in the general XRAM model, but the algorithmic structure and the topology of the interconnection network are taken into consideration by the following *locality assumption*:

*A processor $P_i$ can execute a task $T_j$ if and only if all the predecessors of $T_j$ have been executed either on $P_i$ or on neighbor processors.*

Remark that this execution model can be used efficiently for many families of precedence graphs (precedence graphs of small bounded degree on linear or grid networks, etc.).

#### 5.4.1. Application to trees and grids

Complete binary trees and 2D-grids can be executed optimally in most of the popular interconnection networks (namely, rings, torus, hypercubes and de Bruijn graphs [1]) [8].

[1] For a formal definition of all these networks, see [13].

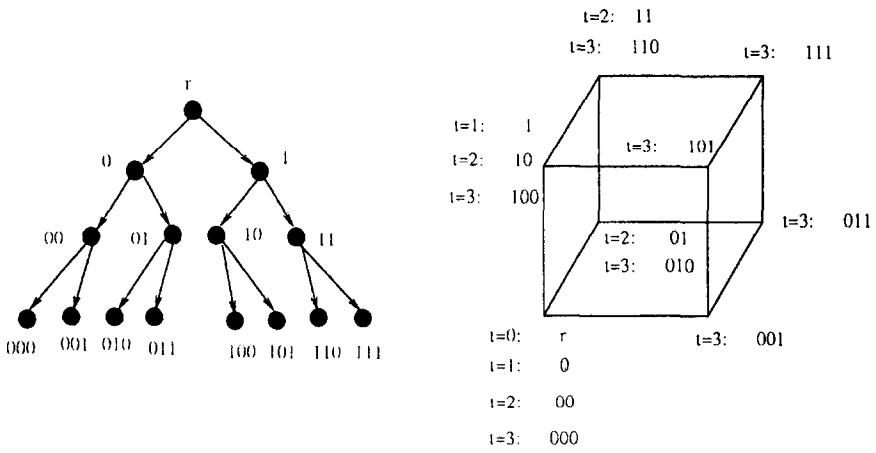



Fig. 8. Execution of the complete binary tree of depth $h = 3$. $\mathscr{T}_3$ on the hypercube $Q_3$.

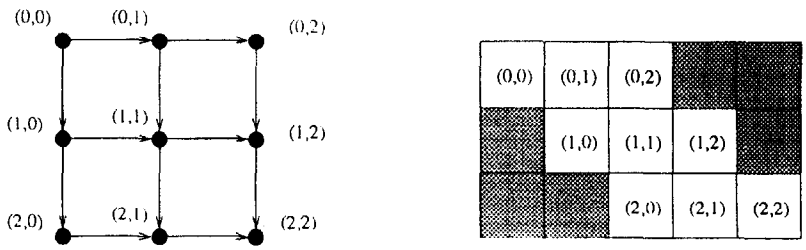



Fig. 9. Optimal execution of a 2D-grid on a linear network with 3 processors.

In Fig. 8, we illustrate the optimal algorithm for the execution of a complete binary tree of depth $h = 3$ on the hypercube $Q_3$. In Fig. 9, we give an example with the execution of a 2D-grid, on a linear network with 3 processors [8]. It is easy to see that an algorithm which executes as soon as possible in a same processor all the tasks with the same first coordinate, respects the locality assumption and leads to the optimal result.

Tables 5 and 6 summarize the known results for the computation of complete binary trees and grids, respectively, into most popular interconnection networks.

## 6. A synthetic view of the different models

Several models have been presented in this paper, and some remarks and relations can be pointed out:

- For the models which do not allow the overlap of the communications by the computations (ABR, ACS, PU), the number of communications is a critical criterion, implying the choice of this number as objective function or a part of it. Concerning the models allowing overlapping, this criterion has not the same importance since the

Table 5
Lower and upper bounds for the execution of complete binary trees into most popular interconnection networks

| Topology | Complete binary trees of depth $h$ [8] | |
|---|---|---|
| | Lower bound | Upper bound |
| Hypercube $Q_n$ | $h+1$ if $n \geq h$<br>$n-1+2^{h-n+1}$ when $h > n$ | $h+1$ if $n \geq h$<br>$n-1+2^{h-n+1}$ when $h > n$ |
| de Bruijn $UB(2,n)$ | $h+1$ if $n \geq h$<br>$n-1+2^{h-n+1}$ when $h > n$ | $h+1$ if $n \geq h$<br>$n-1+2^{h-n+1}$ when $h > n$ |
| Linear networks | $\frac{2^h}{h} + O(2^h/h^2)$ | $\frac{2^h}{h} + O(2^h/h^2)$ |
| Grids | $\frac{2^h}{h^2} + O(2^h/h^3)$ | Open for $h > 6$ |

Table 6
Lower and upper bounds for the execution of 2D-grids of size $(n_1, n_2)$

| Topology | 2D-grids $(n_1, n_2)$ [8] | |
|---|---|---|
| | Lower bound | Upper bound |
| Networks with more than $n$ processors | $n_1 + n_2 - 1$ | $n_1 + n_2 - 1$ |

main goal becomes the minimization of the makespan. This remark leads to different choices of scheduling strategies, according to the possibility to overlap or not the communications.

- Let us now consider the models that do not allow overlap, i.e. ABR, ACS, PU. The scheduling strategy cannot be exactly the same within all these models. Indeed, the strategy of gathering used in [2] cannot be used for the ACS model because of synchronization. However, it is possible to take advantage of gathering for the ACS model, for instance by gathering the UET tasks of fork-type (or join-type) DAGs, in order to avoid a lot of writes (reads) in (from) the shared memory. On the other hand, gathering seems to be very useful for the PU model, since a "nice" gathering can decrease both the number of communications for a given path and the total amount of communications in the schedule.
- We can remark that a valid schedule given in the RS model is also valid within the PY model (for $\tau = 1$). The contrary is not true because of the possibility of using duplication and an unbounded number of processors in the PY model.
- In [13] the authors relate the PY model with the PU model. They proved that if there is a schedule, say $S$, with computation time $T_{\text{comput}}$ and communication delay $T_{\text{delay}}$ within the PU model, then there is a schedule $S'$ such that the makespan of $S'$ within the PY model is equal to $T_{\text{comput}} + T_{\text{delay}}\tau$.

Table 7
Principal characteristics of the presented models

| Model | Dupl$^{tion}$ | Overlap | Objective to minimize | Number of Processors | Communication (value or number) |
|---|---|---|---|---|---|
| ABR | No | No | $T_{com} + T_{idle}$ | Fixed | 0 (if same job)<br>$c$ otherwise |
| PU | Allowed | No | com$^{tion}$ traffic and/or delay | Unbounded or fixed | 0 (same processor)<br>1 otherwise |
| PY | Allowed | Allowed | Makespan | Unbounded | 0 (same processor)<br>$\tau$ otherwise |
| RS | No | Allowed | Makespan | Not fixed | 0 (same processor)<br>1 otherwise |
| ACS | No | No | Computation time and com$^{tion}$ steps | Fixed | 0 (same processor)<br>1 otherwise |
| BKT | No | Allowed | Makespan | Not fixed | 0 but local execution |

- Given a schedule within the ABR model, this schedule can be transformed easily to a valid schedule for the PU model (under the assumption of a fixed number of processors). For each job, one unit of time has to be removed to obtain $T_{comput}$. We can then calculate $T_{delay}$ by counting the number of jobs crossed by any path of the schedule graph minus one (if a path crosses two jobs then it contains only one communication arc). The opposite is false because of the duplication.

Table 7 summarizes some observations concerning the models that we have considered in this paper.

## References

[1] F. Afrati, C.H. Papadimitriou and G. Papageorgiou, Scheduling DAGs to minimize time and computation, in: Proceedings of the Aegian Workshop on Computing (AWOC) (1988) 134–138.
[2] R.J. Anderson, P. Beame and W. Ruzzo, Low overhead parallel schedules for task graphs, in: Proceedings of SPAA (1990) 66–75.
[3] A. Aggarwal, A.K. Chandra and M. Snir, Communication complexity of PRAMs, Theoret. Comput. Sci. 71 (1990) 3–28.
[4] E. Bampis, L'impact des communications sur la complexité des algorithmes parallèles, Ph.D. Thesis. University of Paris-Sud, France (1993).
[5] E. Bampis, C. Delorme and J-C. König, Optimal schedules for dD-grid graphs with communication delays, in: C. Puech, R. Reischuk eds., STACS'96 LNCS No. 1046, 655–666.
[6] E. Bampis, J-C. König and D. Trystram, Impact of communications on the complexity of the parallel Gaussian elimination, Parallel Comput. 17 (1991) 55–61.
[7] E. Bampis, J-C. König and D. Trystram, A low overhead schedule for a 3D-Grid Graph, Parallel Process. Lett. 2 (1992) 363–372.
[8] E. Bampis, J-C. König and D. Trystram, Optimal parallel execution of complete binary trees and grids into most popular interconnection networks, Theoret. Comput. Sci. 147 (1995) 1–18.
[9] E. Bampis, J.-C. König and D. Trystram, Minimizing the schedule length for a parallel 3D-grid precedence graph, European J. Oper. Res., to appear.
[10] E.G. Coffman and P.J. Denning, Operating Systems Theory (Prentice Hall, Englewood Cliffs, NJ, 1972).
[11] J.-Y. Colin and P. Chretienne, CPM scheduling with small interprocessor communication delays, Oper. Res. 39 (1991) 680–684.

[12] M. Cosnard and A. Ferreira, Designing parallel non numerical algorithms, in: D.J. Evan et al., eds., Parallel Computing '91, (North-Holland, Amsterdam, 1991) 3–18.
[13] J. DE Rumeur, Communications dans les réseaux de processeurs, Masson, collection ERI, 1994.
[14] L. Gao, A.L. Rosenberg and R.K. Sitaraman, Optimal architecture-independent scheduling of fine-grain tree-sweep computations, in: Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing (1995) 620–629.
[15] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness (Freeman, New York, 1979).
[16] F. Guinand, Ordónnancement avec communications pour architectures multiprocesseurs dans divers modèles d'exécution, Ph.D. Thesis, INPG, Grenoble, France (1995).
[17] F. Guinand and D. Trystram, Optimal scheduling of UECT trees on two processors, Technical Report APACHE, $n^\circ$ 3, LMC-IMAG, Grenoble, France (1993).
[18] A. Jakoby and R. Reischuk, The complexity of scheduling problems with communication delays for trees, in: Proceedings of the Scandinavian Workshop of Algorithmic Theory, Lecture Notes in Computer Science, Vol. 621 (Springer, Berlin, 1992) 165–177.
[19] H. Jung, L. Kirousis and P. Spirakis, Lower bounds and efficient algorithms for multiprocessor scheduling of DAGs with communication delays, in: Proceedings of the ACM Symposium on Parallel Algorithms and Architectures (1989) 254–264. Inform. Comput. 105 (1993) 94–104.
[20] B.J. Lageweg, J.K. Lenstra and B. Veltman, Multiprocessor scheduling with communication delays, Parallel Comput 16 (1990) 173–182.
[21] E.L. Lawler, Scheduling trees on multiprocessors with unit communication delays, Workshop on Models and Algorithms for Planning and Scheduling Problems, Villa Vigoni, Lake Como, Italy, 14–18 June (1993).
[22] T. Leighton, Methods for message routing in parallel machines, in: Proceedings of the 24th Annual ACM Symposium on Theory Comput. (1992) 77–96.
[23] M.G. Norman, P. Thanisch and K.-F. Chang Partitioning DAG computations: a cautionary note, in: W. Joosen and E. Milgrom, eds., Parallel Computing: From Theory to Sound Practice (IOS Press, 1992) 360–363.
[24] C. Papadimitriou and J. Ullman, A communication time tradeoff, SIAM J. Comput. 16 (1987) 639–646.
[25] C. Papadimitriou and M. Yannakakis, Towards an architecture-independent analysis of parallel algorithms. in: Proceedings of 20th ACM Symposium on Theory of Computing (1988). SIAM J. Comput. 19 (1990) 322–328.
[26] C. Picouleau, Elude des Problèmes d'Optimisation dans les Systèmes Distribués, Ph.D. Thesis, University of Paris VI, France (1993).
[27] V.J. Rayward-Smith, UET scheduling with unit interprocessor communication delays, Discrete Appl. Math. 18 (1987) 55–71.
[28] T. Varvarigou, V.P. Roychowdhury and T. Kailath, Scheduling in and out forests in the presence of communication delays. in: Proceedings of International Parallel Processing Symp. (IPPS'93) (1993) 173–182.
[29] T. Varvarigou, V.P. Roychowdhury and T. Kailath, E. Lawler, Scheduling in and out forests in the presence of communication delays, IEEE Tran. Parallel Distributed Systems, to appear.
[30] M. Veldhorst, A linear time algorithm to schedule trees with communication delays optimally on two machines. Technical Report RUU-CS-93-04, Department of Computer Science, Utrecht (January 1993).
[31] B. Veltman, Multiprocessor scheduling with communication delays, Ph.D. Thesis, CWI-Amsterdam, Holland (1993).
[32] B. Veltman, B.J. Lageweg and J.K. Lenstra, Multiprocessor scheduling with communication delays, Parallel Comput. 16 (1990) 173–182.