# Linear-time Approximation Schemes for Scheduling Malleable Parallel Tasks

Klaus Jansen　　Lorant Porkolab

MPI–I–98–1-025　　October 1998

## Author's Address

Klaus Jansen
IDSIA Lugano
Corso Elvezia 36
6900 Lugano
Switzerland
klaus@idsia.ch

Lorant Porkolab
Max-Planck-Institute for Computer Science
Im Stadtwald
66123 Saarbrücken
Germany
porkolab@mpi-sb.mpg.de

## Acknowledgements

Research supported in part by EU ESPRIT LTR Project No. 20244 (ALCOM-IT) and by the Swiss Office Fédéral de l'éducation et de la Science project n 97.0315 titled "Platform".

## Abstract

A malleable parallel task is one whose execution time is a function of the number of (identical) processors alloted to it. We study the problem of scheduling a set of $n$ independent malleable tasks on a fixed number of parallel processors, and propose an approximation scheme that for any fixed $\epsilon > 0$, computes in $O(n)$ time a non-preemptive schedule of length at most $(1 + \epsilon)$ times the optimum.

## 1 Introduction

In this paper, we study the following scheduling problem. Suppose there is given a set of tasks $\mathcal{T} = \{T_0, \ldots, T_{n-1}\}$ and a set of identical processors M $= \{1, \ldots, m\}$. Each task $T_j$ has an associated function $t_j : M \to Q^+$ that gives the execution time $t_j(\ell)$ of task $T_j$ in terms of the number of processors $\ell \in M$ that are assigned to $T_j$. Given $\beta_j$ processors alloted to task $T_j$, these $\beta_j$ processors are required to execute task $T_j$ in union and without preemption, i.e. they all have to start processing task $T_j$ at some starting time $\tau_j$, and complete it at $\tau_j + t_j(\beta_j)$. A feasible non-preemptive schedule consists of a processor allotment $\beta_j \in M$ and a starting time $\tau_j \geq 0$ for each task $T_j$ such that for each time step $\tau$, the number of active processors does not exceed the total number of processors, i.e.

$$\sum_{j:\tau \in [\tau_j, \tau_j + t_j(\beta_j))} \beta_j \;\leq\; m.$$

The objective is to find a feasible non-preemptive schedule that minimizes the overall makespan

$$\max\{\tau_j + t_j(\beta_j) : j = 0, \ldots, n-1\}.$$

This problem is called malleable parallel task scheduling (MPTS), and has recently been studied in several papers, see e.g. [3, 7, 16, 19]. The problem of non-malleable parallel task scheduling (NPTS) is a restriction of MPTS in which the processor allotments are known a priori, i.e. for each task both the number of assigned processors and its execution time are given as part of the input. Closely related problems to NPTS are rectangle packing (see e.g. [4, 6, 13, 18]) and resource constrained scheduling (see e.g. [5, 12]).

In several applications, a network topology is also specified for the processors (see e.g. [8, 9, 14, 16]). In these cases the tasks are scheduled not on an arbitrary subset of processors, but on a subset of processors with a particular interconnection network that depends on the underlying architecture. For example on hypercubes and meshes, the tasks would require subcubes and submeshes, respectively. If the underlying architecture is not taken into account, the problem is referred to as scheduling on a PRAM. For simplicity, we will consider only the latter problem in this paper, but the results can also be extended to arbitrary network topologies (e.g. lines, hypercubes and meshes).

The NPTS and MPTS problems are strongly NP-hard [7], and 2 is the best currently known approximation ratio for them achieved in polynomial time [10, 16]. For a particular input $I$, let $OPT(I)$ be the minimum makespan, and let $A(I)$ denote the makespan obtained by algorithm $A$. A polynomial-time approximation scheme for this problem is an algorithm $A$, which for any (constant) $\epsilon > 0$ and input $I$ outputs in time polynomial in the length of $I$ a feasible schedule $A(I)$ with performance guarantee $R_A(I, \epsilon) = \frac{A(I)}{OPT(I)} \leq 1+\epsilon$. Such an algorithm can also be viewed as a family of algorithms $\{A_\epsilon | \epsilon > 0\}$ such that $A_\epsilon(I) \leq (1+\epsilon)OPT(I)$. A fully polynomial approximation scheme is an approximation scheme $A$ that runs in time polynomial not only in the length of $I$ but also in $\frac{1}{\epsilon}$. Regarding the problems discussed in this paper, the existence of an approximation scheme was previously known only for NPTS on hypercubes of fixed dimension [14, 15].

Since both NPTS and MPTS are strongly NP-hard even for a fixed number ($m \geq 5$) of processors [7], it is natural to ask how well the optimum for these restricted variants can be approximated. In this paper, we focus on the case when there are only a constant number of processors and present polynomial-time approximation schemes for both MPTS and NPTS which compute for any fixed $\epsilon > 0$ $\epsilon$-approximate schedules in $O(n)$ time.

The main steps of the approximation scheme are the following. First, it computes $d_j = \min_{\ell=1,\ldots,m} t_j(\ell)$ for each task $T_j$ and selects a constant number $k = k(m,\epsilon)$ of tasks $T_{j_1}, \ldots, T_{j_k}$ with the largest $d_j$ values. Next, it constructs all relative schedules for the set $\mathcal{L} = \{T_{j_1}, \ldots, T_{j_k}\}$ consisting of processor assignments and an execution order of the tasks in $\mathcal{L}$. For each relative schedule, there is a (mixed integer) linear program for scheduling all tasks in $\mathcal{T}$ such that the relative schedule of $\mathcal{L}$ is respected. This linear program can be decomposed into two parts: a fractional packing problem and a linear program with a constant number of variables and constraints. By using this decomposition and an approximation scheme for packing problems the algorithm solves the linear programming relaxation approximately. Then based on this solution, it computes a schedule for most of the tasks, and then executes the rest of them at the end. We will show that the makespan of the final (feasible) schedule produced by this procedure is at most $(1+\epsilon)$ times the optimum. The running time of the algorithm is not polynomial in $\frac{1}{\epsilon}$, but this is not surprising, since due to the strong NP-hardness no fully polynomial approximation scheme can be expected.

## 2 Linear Programs

In this section, first we consider the NPTS problem and formulate it as a linear program. A similar LP formulation was given in [1] for scheduling independent multiprocessor tasks on dedicated processors, where each task requires a prespecified subset of processors. Afterwards, we describe how the linear program for NPTS can be extended to MPTS by introducing new variables and additional constraints. Based on this linear programming formulation we will give a linear time approximation scheme for MPTS in Section 3.

**2.1 Non-Malleable Tasks.** For each task $T_j \in \mathcal{T}$, let the number of allotted processors be denoted by $\beta_j$ and the execution time by $p_j$. Let $\mathcal{L} \subset \mathcal{T}$. A processor assignment for $\mathcal{L}$ is a mapping $f : \mathcal{L} \rightarrow 2^M$ such that $|f(T_j)| = \beta_j$ for each task $T_j \in \mathcal{L}$. Two tasks $T_j$ and $T_{j'}$ are compatible, if $f(T_j) \cap f(T_{j'}) = \emptyset$. For a given processor assignment for $\mathcal{L}$, a *snapshot* of $\mathcal{L}$ is a subset of compatible tasks. A *relative schedule* of $\mathcal{L}$ is a processor assignment $f : \mathcal{L} \rightarrow 2^M$ along with a sequence $M(1), \ldots, M(g)$ of snapshots of $\mathcal{L}$ such that

- each $T_j \in \mathcal{L}$ occurs in a subsequence of consecutive snapshots $M(\alpha_j), \ldots, M(\omega_j)$, $1 \leq \alpha_j \leq \omega_j < g$, where $M(\alpha_j)$ is the first and $M(\omega_j)$ is the last snapshot that contains $T_j$;

- consecutive snapshots are different, i.e. $M(t) \neq M(t+1)$ for $1 \leq t \leq g-1$;
- $M(1) \neq \emptyset$ and $M(g) = \emptyset$.

A relative schedule corresponds to an order of executing the tasks in $\mathcal{L}$. One can associate a relative schedule for each non-preemptive schedule of $\mathcal{L}$ by looking at the schedule at every time where a task of $\mathcal{L}$ starts or ends and creating a snapshot right after that time step. Creating snapshots this way, $M(1) \neq \emptyset$, $M(g) = \emptyset$ and the number of snapshots can be bounded by $\max(2|\mathcal{L}|, 1)$. Given a relative schedule $R = (f, M(1), \ldots, M(g))$, the processor set used in snapshot $M(i)$ is given by $P(i) = \cup_{T \in M(i)} f(T)$. Let $\mathcal{F}$ denote the set containing (as elements) all the different $(M \setminus P(i))$ sets, $i = 1, \ldots, g$. (Thus the sets in $\mathcal{F}$ are the different sets of free processors corresponding to $R$.) For each $F \in \mathcal{F}$, let $P_{F,i}, i = 1, \ldots, n_F$, denote the different partitions of $F$, and let $\mathcal{P}_F = \{P_{F,1}, \ldots, P_{F,n_F}\}$. Furthermore let $D^\ell$ be the total processing time for all tasks in $\mathcal{S} = \mathcal{T} \setminus \mathcal{L}$ executed on $\ell$ processors. For each partition $P_{F,i}$, the number of processor sets $F_j \in P_{F,i}$ with cardinality $\ell$ is denoted by $a_\ell(F, i)$. (These $F_j$'s are reserved for $\ell$-processor tasks, i.e. for tasks that use exactly $\ell$ processors.) The number $a_\ell(F, i)$ gives the parallelization factor of $\ell$-processor tasks for partition $P_{F,i}$.

For each relative schedule $R = (f, M(1), \ldots, M(g))$ of $\mathcal{L}$, we formulate a linear program $LP(R)$, as follows.

**Minimize** $t_g$ **s.t.**

**(0)** $t_0 = 0,$

**(1)** $t_i \geq t_{i-1}, \quad i = 1, \ldots, g,$

**(2)** $t_{\omega_j} - t_{\alpha_j - 1} = p_j, \quad \forall T_j \in \mathcal{L},$

**(3)** $\sum_{i:P(i)=M\setminus F} (t_i - t_{i-1}) = e_F, \quad \forall F \in \mathcal{F},$

**(4)** $\sum_{i=1}^{n_F} x_{F,i} \leq e_F, \quad \forall F \in \mathcal{F},$

**(5)** $\sum_{F\in\mathcal{F}} \sum_{i=1}^{n_F} a_\ell(F, i) \cdot x_{F,i} \geq D^\ell, \quad \ell = 1, \ldots, m,$

**(6)** $x_{F,i} \geq 0, \quad \forall F \in \mathcal{F}, \ i = 1, \ldots, n_F.$

The variables of $LP(R)$ have the following interpretation:

$t_i$: the time when snapshot $M(i)$ ends (and $M(i+1)$ starts), $i = 1, \ldots, g-1$. The starting time of the schedule and snapshot $M(1)$ is denoted by $t_0 = 0$ and the finishing time by $t_g$.

$e_F$: the total time while exactly the processors in $F$ are free.

$x_{F,i}$: the total processing time for $P_{F,i} \in \mathcal{P}_F$, $i = 1, \ldots, n_F, F \in \mathcal{F}$, where only processors of $F$ are executing short tasks and each subset of processors $F_j \in P_{F,i}$ executes at most one short task at each time step in parallel.

The given relative schedule $R$ along with constraints (1) and (2) define a feasible schedule of $\mathcal{L}$. In (3), the total processing times $e_F$, for all $F \in \mathcal{F}$ are determined. Clearly, these equalities can be inserted directly into (4). The inequalities in (4) require for every set of free processors $F \in \mathcal{F}$ that its total processing time (corresponding to the different partitions) to be bounded by $e_F$. Furthermore, the inequalities (5) guarantee that there is enough time for the execution of all $\ell$-processor tasks in $\mathcal{S}$.

Notice that the solutions of $LP(R)$ allow for each $\ell$-processor task from $\mathcal{S}$: to be preempted, to be executed in parallel on multiple subsets of processors from $P_{F,i}$ of cardinality $\ell$, to change processor assignments during the execution. Thus there might be incorrectly scheduled tasks in the schedule based on the solution of $LP(R)$. These have to be corrected afterwards.

**2.2 Malleable Tasks.** In this section, we show how the above linear program can be extended for MPTS. Suppose that the different processing times of task $T_j \in \mathcal{T}$ are given by the function $t_j : M \to Q^+$. In the NPTS problem, the number of alloted processors for each task is known a priori, therefore the $D^\ell$ values in $LP(R)$ are fixed constants. However in the MPTS problem, each task can be executed on an arbitrary number of available processors, and therefore the $D^\ell$'s cannot be considered fixed anymore. Note that (for a fixed relative schedule $R$) all the other coefficients (and constraints) of system (0) – (6) are independent from the processor allotments, thus they remain the same for MPTS. In order to handle the possibility of non-fixed processor allotments, we introduce $0-1$-variables $y_{j\ell}$ for each task $T_j \in \mathcal{S}$ and each number $\ell \in M$ with the interpretation that

$$y_{j\ell} = \begin{cases} 1, & \text{if } T_j \text{ is executed on } \ell \text{ processors,} \\ 0, & \text{otherwise.} \end{cases}$$

In fact, these variables will be relaxed later. For a given relative schedule $R$, let $ILP(R)$ denote the extension of $LP(R)$ to MPTS. In $ILP(R)$, the $D^\ell$'s, $\ell = 1, \ldots, m$, are variables (in contrast to $LP(R)$, where they are constant coefficients), and we have the following constraints in addition to (0) – (6):

**(7)** $\sum_{T_j \in \mathcal{S}} t_j(\ell) \cdot y_{j\ell} = D^\ell, \quad \ell = 1, \ldots, m,$

**(8)** $\sum_{\ell=1}^{m} y_{j\ell} = 1, \quad \forall T_j \in \mathcal{S},$

**(9)** $y_{j\ell} \in \{0, 1\}, \quad \forall T_j \in \mathcal{S}, \ \ell = 1, \ldots, m.$

The constraints of (8) – (9) describe the processor allotments for the tasks in $\mathcal{S}$. The equations of (7) express for every $\ell = 1, \ldots, m$, $D^\ell$, the total processing time of all tasks in $\mathcal{S}$ that are executed on $\ell$ processors. As before, these equations can be inserted directly into the inequalities of (5).

**2.3 Relaxation.** As it was pointed out above, for any fixed relative schedule, the only difference between NPTS and MPTS with respect to system (0) – (6) is that the $D^\ell$'s are not constants any more. Therefore the MPTS problem can also be solved by solving $LP(R)$ for every relative schedule $R$ (of MPTS) and every possible $m$-vector $(D^1, \ldots, D^m)$, and then selecting the best solution. However generating all of the $m^{|\mathcal{S}|}$ possible $(D^1, \ldots, D^m)$ vectors (e.g. by considering all of the $0-1$ feasible solutions of (8) – (9)) require an exponential number of operations in terms of $n$. To avoid the exponential dependence on $n$ (recall that our goal is to solve the problem in $O(n)$ time), we will use the following relaxation of (7) – (9) to compute only all "approximately different" $m$-vectors $(D^1, \ldots, D^m)$. (This procedure will be described in Section 3.3 in detail.)

**(7)'** $\sum_{T_j \in \mathcal{S}} t_j(\ell) \cdot y_{j\ell} \ \le \ D^\ell, \quad \ell = 1, \ldots, m,$

**(8)'** $\sum_{\ell=1}^{m} y_{j\ell} = 1, \quad \forall T_j \in \mathcal{S},$

**(9)'** $y_{j\ell} \ge 0, \quad \forall T_j \in \mathcal{S},\ \ell = 1, \ldots, m.$

Let $ELP(R)$ denote the linear program consisting of $LP(R)$ along with $(7)' - (9)'$. Notice, that for any relative schedule $R$, $ELP(R)$ is a relaxation of $ILP(R)$, and therefore the optimum of $ELP(R)$ is a lower bound on the minimum makespan for schedules respecting $R$. Also note that in the above constraints the new $y_{j\ell}$ variables are not assumed to be integers, and therefore in general they attain some fractional values in the solution of $ELP(R)$.

## 3 Algorithm

In this section, we present an approximation scheme for the MPTS problem. First, we describe the main algorithm and then discuss some of the steps in detail. The procedure is based on selecting a small subset $\mathcal{L} \subset \mathcal{T}$ with cardinality $k = k(m, \epsilon)$. In the following we assume that $n > k = k(m, \epsilon)$, since otherwise one can easily compute an optimal solution in constant$(m, \epsilon)$ time by considering all feasible schedules for $\mathcal{T}$.

### 3.1 The Main Algorithm.

1. Compute $d_j = \min_{1 \le \ell \le m} t_j(\ell)$, $j = 0, \ldots, n-1$, and assume that $d_j = t_j(\phi_j)$, for some $\phi_j \in M$. Set $D = \sum_{j=0}^{n-1} d_j$, $\mu = \frac{\epsilon}{2m}$, and $K = 4m^{m+1}(2m)^{\lceil \frac{1}{\mu} \rceil + 1}$.

2. Select the $K+1$ longest tasks with respect to the $d_j$'s. Assume $d_0 \ge d_1 \ge \ldots \ge d_K$. Find the smallest $k \le K$ such that $d_k + \ldots + d_{2mk+3m^{m+1}-1} \le \mu \cdot D$. Partition the set of tasks $\mathcal{T}$ into two subsets $\mathcal{L}$ and $\mathcal{S}$ such that $\mathcal{L}$ contains the $k$ longest tasks according to the $d_j$'s.

3. Construct all possible relative schedules of $\mathcal{L}$ consisting for each task $T_j \in \mathcal{L}$ a processor allotment $\beta_j$, a processor assignment $f(T_j) \subseteq M$ of cardinality $\beta_j$, and a sequence of snapshots $M(1), \ldots, M(g)$ for $\mathcal{L}$.

4. For each relative schedule $R$ of $\mathcal{L}$:

   4.1. Compute an approximate solution $(t, e, x, y)$ of $ELP(R)$ by (approximately) solving $LP(R)$ for every "approximately different" $m$-vector $(D^1, \ldots, D^m)$ for which the packing problem $(7)' - (9)'$ is feasible. (See Section 3.3.)

   4.2. Convert the assignment part $y$ into an equivalent one represented by a forest and determine the set $\mathcal{V}$ of those tasks for which it gives a non-unique processor allotment. (See Section 3.4.)

   4.3. Construct a pseudo-schedule $S(f, M)$ for the tasks in $\mathcal{S} \setminus \mathcal{V}$. (See Section 3.5.)

   4.4. Determine the set $\mathcal{U}$ of those tasks which are scheduled incorrectly in $S(f, M)$. Then, modify $S(f, M)$ by executing sequentially every $T_j \in \mathcal{V} \cup \mathcal{U}$ at the end of the schedule using $\phi_j$ processors.

5. From all schedules constructed in step 4.4, select one that has the smallest makespan.

**3.2 Example for MPTS.** Consider a subset of tasks $\mathcal{L} = \{1, 2, 3\}$ with processor assignments $f(1) = \{2\}$, $f(2) = \{3\}$, $f(3) = \{1\}$ and execution times $p_1 = 12$, $p_2 = 15$ and $p_3 = 25$. A relative schedule $R$ for $\mathcal{L}$ is given in Figure 1. In this relative schedule the sequence of snapshots is

$$(\{1,3\}, \{3\}, \{2,3\}, \{2\}, \emptyset)$$

with processor sets $P(1) = \{1, 2\}$, $P(2) = \{1\}$, $P(3) = \{1, 3\}$, $P(4) = \{3\}$ and $P(5) = \emptyset$. The execution times of the other tasks in $\mathcal{T} \setminus \mathcal{L}$ are given in Table 1.

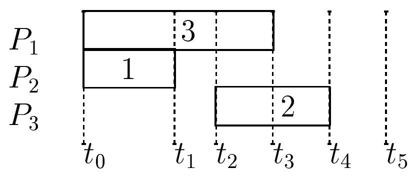

Figure 1: Relative schedule $R$ for tasks $1, 2, 3$.

| task $T_j$ | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|
| $t_j(1)$ | 5 | 6 | 6 | 4 | 10 | 10 | 10 |
| $t_j(2)$ | 3 | 2 | 2 | 2 | 5 | 6 | 7 |
| $t_j(3)$ | 2 | 1 | 1 | 1 | 2 | 4 | 5 |

Table 1: Execution times for tasks in $\mathcal{T} \setminus \mathcal{L}$.

The free processor sets in this particular relative schedule are $\{3\}, \{2,3\}, \{2\}, \{1,2\}, \{1,2,3\}$, and therefore, the following partitions have to be considered: $P_{\{3\},1} = (\{3\})$, $P_{\{2,3\},1} = (\{2,3\})$, $P_{\{2,3\},2} = (\{2\},\{3\})$, $P_{\{2\},1} = (\{2\})$, $P_{\{1,2\},1} = (\{1,2\})$, $P_{\{1,2\},2} = (\{1\},\{2\})$, $P_{\{1,2,3\},1} = (\{1,2,3\})$, $P_{\{1,2,3\},2} = (\{1\},\{2,3\})$, $P_{\{1,2,3\},3} = (\{2\},\{1,3\})$, $P_{\{1,2,3\},4} = (\{3\},\{1,2\})$, $P_{\{1,2,3\},5} = (\{1\},\{2\},\{3\})$.

For the given relative schedule $R$ of Figure 1, the linear program $ELP(R)$ is the following:

**Minimize** $t_5$ **s.t.**

$$t_1 = 12,\ t_2 \geq t_1,\ t_3 = 25,\ t_3 \geq t_2,\ t_4 = t_2 + 15,\ t_4 \geq t_3,\ t_5 \geq t_4,$$

$$e_{\{3\}} = 12,\ e_{\{2,3\}} = t_2 - t_1,\ e_{\{2\}} = t_3 - t_2,\ e_{\{1,2\}} = t_4 - t_3,\ e_{\{1,2,3\}} = t_5 - t_4,$$

$$x_{\{3\},1} \leq e_{\{3\}},\ \ldots,\ x_{\{1,2,3\},1} + \ldots + x_{\{1,2,3\},5} \leq e_{\{1,2,3\}},$$

$$x_{\{3\},1} + 2x_{\{2,3\},2} + \ldots + 3x_{\{1,2,3\},5} \geq D^1,\ \ldots,\ x_{\{1,2,3\},1} \geq D^3,$$

$$x_{\{3\},1}, \ldots, x_{\{1,2,3\},5} \geq 0,$$

$$5y_{4,1} + 6y_{5,1} + \ldots + 10y_{10,1} \leq D^1,\ \ldots,\ 2y_{4,3} + y_{5,3} + \ldots + 5y_{10,3} \leq D^3,$$

$$y_{4,1} + y_{4,2} + y_{4,3} = 1,\ \ldots,\ y_{10,1} + y_{10,2} + y_{10,3} = 1,$$

$$y_{4,1}, \ldots, y_{10,3} \geq 0.$$

In a solution of this linear program, all tasks have a unique processor allotment: $\beta_4 = 1$, $\beta_5 = 3$, $\beta_6 = 3$, $\beta_7 = 1$, $\beta_8 = 3$, $\beta_9 = 1$ and $\beta_{10} = 1$ (clearly, this does not hold in general). The assignment of the tasks in $\mathcal{T} \setminus \mathcal{L}$ to the free processors is illustrated in Figure 2. We note that $t_2 = t_1$ and that tasks 9 and 10 are scheduled incorrectly. For task 9 there is a change in processor assignment at time $t_1$, and task 10 with allotment $\beta_{10} = 1$ is executed on two processors during the interval $[t_3, t_4)$. At step 4.4 of the algorithm, both tasks are removed and scheduled sequentially at the end of the schedule after time $t_5$ on three processors.

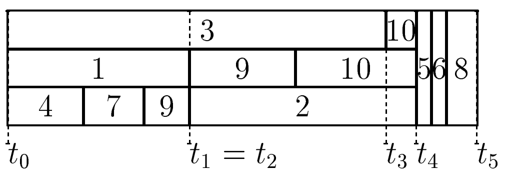

Figure 2: Pseudo-schedule for all tasks in $\mathcal{T}$.

**3.3 Approximate Solution for ELP(R).** Consider now the packing problem $P(d)$ defined by constraints $(7)' - (9)'$ for a given $m$-vector $d = (D_1, \ldots, D_m)$. This problem has a special block angular structure, where the blocks $B_j = \{y_j \in R^m \mid y_{j\ell} \geq 0,\ \sum_{\ell=1}^m y_{j\ell} = 1\}$, for $T_j \in \mathcal{S}$, are $m$-dimensional simplicies, and the coupling constraints are the linear inequalities $\sum_{T_j \in \mathcal{S}} t_j(\ell) \cdot y_{j\ell} \leq D^\ell$, $\ell = 1, \ldots, m$. The Logarithmic Potential Price Directive Decomposition Method [11] developed for a large class of problems with block angular structure provides a $\rho$-relaxed decision procedure for $P(d)$. This procedure either determines that $P(d)$ is infeasible, or computes (a solution that is nearly feasible in the sense that it is) a feasible solution of $P((1+\rho)d)$. This can be done (see Theorem 3 of [11]) in $m(\ln m + \rho^{-2} \ln \rho^{-1})$ iterations, where each iteration requires $O(m \ln\ln(m\rho^{-1}))$ operations and $|\mathcal{S}| \leq n$ block optimizations performed to a relative accuracy of $O(\rho)$. In our case each block optimization is the minimization of a given linear function over an $m$-dimensional simplex which can be done (not only approximately, but even) exactly in $O(m)$ time. Therefore the overall running time of the procedure for $P(d)$ is $O([n + \ln\ln\frac{m}{\rho}](\frac{m}{\rho})^2 \ln\frac{m}{\rho})$.

The next lemma provides lower and upper bounds for $OPT$, the optimum makespan of the MPTS problem, in terms of the minimum execution times.

Lemma 3.1. *Let* $d_j = \min_{1 \leq \ell \leq m} t_j(\ell)$, $j = 0, \ldots, n-1$, *and* $D = \sum_{j=0}^{n-1} d_j$. *Then* $D \geq OPT \geq \frac{D}{m}$.

*Proof.* Consider an optimum solution with schedule length $OPT$, processor allotments $\beta_j \geq 1$ and execution times $t_j(\beta_j)$. Then $d_j \leq t_j(\beta_j)$, and thus we obtain by using an averaging argument that

$$OPT \geq \sum_{j=0}^{n-1} \frac{\beta_j \cdot t_j(\beta_j)}{m} \geq \frac{1}{m} \cdot \sum_{j=0}^{n-1} d_j = \frac{1}{m} \cdot D.$$

□

This lemma implies that every $D^\ell$, $\ell = 1, \ldots, m$, in $ELP(R)$ can also be bounded by $mD/\ell$. Suppose $\kappa = \kappa(m, \epsilon)$ (it will be specified later) is a rational number such that $\frac{m}{\kappa}$ is an integer. Consider the partition of the interval $[0, mD]$ into $\frac{m}{\kappa}$ subintervals $[0, \kappa)$, $[\kappa, 2\kappa)$, ..., $[mD - \kappa, mD]$ of equal size $\kappa$. Let $\Delta = \{\ i\kappa D : i = 0, 1, \ldots, \frac{m}{\kappa}\ \}^m$. The algorithm determines the set $\mathcal{D} \subseteq \Delta$ that contains every $m$-vector $(D^1, \ldots, D^m) \in \Delta$ with $D^\ell \leq mD/\ell$ for which the above described $\rho$-relaxed decision procedure returns an approximate solution. For a given relative schedule $R$ and $m$-vector $d = (D^1, \ldots, D^m)$, let $t_g^*(ELP(R))$ and $t_g^*(LP(R), d)$ denote the optimum of $ELP(R)$ and $LP(R)$ with $d$, respectively.

Lemma 3.2. *For any two non-negative $m$-vectors $a$ and $\omega$, the following inequalities hold*

$$t_g^*(LP(R), a) \leq t_g^*(LP(R), a + \omega) \leq t_g^*(LP(R), a) + \sum_{\ell=1}^m \omega_\ell. \tag{3.1}$$

*Proof.* If $(t, e, x)$ is a solution of $LP(R)$ with $(a + \omega)$, then it is also a feasible solution of $LP(R)$ with $a$. This implies the first inequality. To show the second one assume that $(t, e, x)$ is a solution of $LP(R)$ with $a$, and consider $M \in \mathcal{F}$. (Note that $M \in \mathcal{F}$, since $M(g) = P(g) = \emptyset$, for every relative schedule $R$.) It is easy to check that for every $\ell = 1, \ldots, m$, there exists an index $i_\ell \in \{1, \ldots, n_M\}$ such that $a_\ell(M, i_\ell) \neq 0$, and $i_{\ell'} \neq i_\ell$, for every $\ell' \neq \ell$. (E.g. $i_\ell$ can be selected such that $P_{M,i_\ell} = \{\{1, \ldots, \ell\}, \{\ell{+}1\}, \ldots, \{m\}\}$.) Then one can obtain a feasible solution $(t', e', x')$ of $LP(R)$ with $a + \omega$ by modifying $(t, e, x)$ in the following way. Let $x'_{M,i_\ell} = x_{M,i_\ell} + \omega_\ell$, for every $\ell = 1, \ldots, m$, $e'_M = e_M + \sum_{\ell=1}^m \omega_\ell$, $t'_g = t_g + \sum_{\ell=1}^m \omega_\ell$, and let all of the other components of $(t', e', x')$ be the same as in $(t, e, x)$. Clearly $(t', e', x')$ is a feasible solution of $LP(R)$ with $a + \omega$ whose objective function value is $t_g + \sum_{\ell=1}^m \omega_\ell$, hence the second inequality of (3.1) follows. □

Lemma 3.3. *Let $\kappa = \frac{\epsilon}{4m^2}$ and $\rho = \frac{\epsilon}{4m^2(\log m+1)}$. Suppose for a given relative schedule $R$, $Min_{d \in \mathcal{D}}\ t_g^*(LP(R), d) = t_g^*(LP(R), \bar{d})$. Then the optimum solution of $LP(R)$ with $(1+\rho)\bar{d}$ is an approximate solution of $ELP(R)$ such that*

$$t_g^*(ELP(R)) \;\leq\; t_g^*(LP(R), (1+\rho)\bar{d}) \;\leq\; t_g^*(ELP(R)) + \frac{\epsilon}{2} \cdot OPT. \qquad (3.2)$$

*Proof.* By the definition of $\bar{d}$ and $\mathcal{D}$, any solution of $LP(R)$ with $(1+\rho)\bar{d}$ is a feasible solution of $ELP(R)$, therefore the first inequality above holds. Suppose that in the optimum solution of $ELP(R)$, $(D^1, \ldots, D^m) = (\tilde{D}^1, \ldots, \tilde{D}^m) = \tilde{d}$, i.e. $t_g^*(ELP(R)) = t_g^*(LP(R), \tilde{d})$. By the definition of $\Delta$ there exists a $\hat{d} = (\hat{D}^1, \ldots, \hat{D}^m) \in \Delta$ such that $\tilde{D}^\ell \leq \hat{D}^\ell \leq \tilde{D}^\ell + \kappa D$, for every $\ell = 1, \ldots, m$. Then by using Lemma 3.1 and 3.2 we obtain that

$$\begin{aligned}
t_g^*(LP(R), (1+\rho)\bar{d}) &\leq t_g^*(LP(R), \bar{d}) + mD\rho \textstyle\sum_{\ell=1}^m \frac{1}{\ell} \\
&\leq t_g^*(LP(R), \hat{d}) + mD\rho(\log m + 1) \\
&\leq t_g^*(LP(R), \tilde{d} + (\kappa D, \ldots, \kappa D)) + m(\log m + 1)D\rho \\
&\leq t_g^*(LP(R), \tilde{d}) + m\kappa D + m(\log m + 1)D\rho \\
&\leq t_g^*(ELP(R)) + m^2(\kappa + (\log m + 1)\rho) \cdot OPT.
\end{aligned}$$

Since $\kappa = \frac{\epsilon}{4m^2}$ and $\rho = \frac{\epsilon}{4m^2(\log m+1)}$, this implies the second inequality of (3.2). □

Determining $\mathcal{D}$ requires the approximate solution of $|\Delta| \leq (\frac{m}{\kappa})^m$ packing problems $P(d)$ with accuracy $\rho = \frac{\epsilon}{4m^2(\log m+1)}$. Therefore according to the argument in the beginning of this section, this can be done in $n(\frac{m}{\epsilon})^{O(m)}$ time. Then by solving $|\mathcal{D}| \leq (\frac{m}{\kappa})^m$ linear programs $LP(R)$ of constant$(m, \epsilon)$ size (with at most $O(K) = m^{O(\frac{m}{\epsilon})}$ variables and constraints) one can find $\bar{d}$ of Lemma 3.3 and compute the corresponding approximate solution of $ELP(R)$ in constant$(m, \epsilon)$ time. Thus the overall running time of Step 4.1 is $O(n)$ for any fixed $m$ and $\epsilon > 0$.

**3.4 Fractional Components.** The $y$-components of the approximate solution of $ELP(R)$ of Lemma 3.3 (determined as a $\rho$-approximate solution of $P(\bar{d})$) can be considered as fractional assignments. Let the lengths of $y$ be defined as $L^\ell = \sum_{T_j \in \mathcal{S}} t_j(\ell) \cdot y_{j\ell}$, $\ell = 1, \ldots, m$. A fractional assignment $y$ can be represented by a bipartite graph $G = (V_1, V_2, E)$, where $V_1$ and $V_2$ correspond to row and column indices of $y$, respectively, and $(j, \ell) \in E$ if and only if $y_{j\ell} > 0$. Any assignment $y$ represented by a bipartite graph $G$ of lengths $L^\ell$, $\ell = 1, \ldots, m$, can be converted in $O(|E| \min\{|V_1|, |V_2|\}) = O(nm^2)$ time into another (fractional) assignment of lengths at most $L^\ell$, $\ell = 1, \ldots, m$, represented by a forest [17] (see Lemma 5.1).

Therefore we can assume that if the above procedure outputs an approximate solution for $P(d)$ then it is a $\rho$-approximate solution represented by a forest. For a fractional assignment $y$, a task $T_j$ has a non-unique processor allotment if there are at least two processor numbers $\ell$ and $\ell'$, $\ell \neq \ell'$, such that $y_{j\ell} > 0$ and $y_{j\ell'} > 0$. Suppose it is given a fractional assignment $y$ represented by a forest $G$, and consider a connected component $C$ of $G$ with $i$ elements in $V_2$. There can be at most $i-1$ tasks with non-unique processor allotment in $C$, since otherwise there would be at least $2i$ edges in an induced subgraph of $C$ or $G$ with $2i$ vertices. This implies the following result.

Lemma 3.4. $|\mathcal{V}| \leq m - 1$.

**3.5 Generating a Schedule.** Step 4.3 of the main algorithm requires the computation of a pseudo-schedule $S(f, M)$ for the tasks in $\mathcal{S} \setminus \mathcal{V}$, in which the tasks are allowed to be preempted and/or be executed parallel on multiple blocks of processors. In this subsection, first we describe an algorithm to compute a pseudo-schedule, and then give an upper bound on the number of incorrectly scheduled tasks.

Let $\hat{D}^\ell$ be the total processing time for all tasks in $\mathcal{S} \setminus \mathcal{V}$ assigned to $\ell$ processors, and let $(t^*, e^*, x^*)$ be the optimum solution of the corresponding linear program. First, we order all $\ell$-processor tasks in $\mathcal{S} \setminus \mathcal{V}$: $T_{\ell,1}, \ldots, T_{\ell,n_\ell}$, for every $\ell = 1, \ldots, m$. Then, we analyze all intervals $[t^*_a, t^*_{a+1})$ with $t^*_a < t^*_{a+1}$ one after another starting with $[t^*_0 = 0, t^*_1)$. Each interval $[t^*_a, t^*_{a+1})$ has an associated processor set $P(a+1)$ used by the tasks in $\mathcal{L}$. Let $L_{a+1}$ be the length of the interval. For this interval, we consider all different partitions $P_{F,1}, \ldots, P_{F,n_F}$ of the set $F = M \setminus P(a+1)$ (the free processors) with $x^*_{F,i} > 0$. We choose the first partition $P_{F,i}$ with $x^*_{F,i} > 0$ (possibly a part of $x^*_{F,i}$ has been considered before). Then, we select for each set $X \in P_{F,i}$ the first non-completely scheduled task $T_{\ell,j}$ that needs $\ell = |X|$ processors. The processor set $X$ is filled with the tasks $T_{\ell,j}, T_{\ell,j+1}, \ldots$ until the total processing time of the selected tasks becomes at least $L^* = \min\{x^*_{F,i}, L_{a+1}\}$, or until all $\ell$-processor tasks have been scheduled. If the total processing time is strictly greater than $L^*$, then the last selected task is preempted such that the total time is exactly $L^*$. After the assignments of the tasks for the sets $X \in P_{F,i}$, we compare the length $x^*_{F,i}$ of the partition $P_{F,i}$ with the length of the interval $L_{a+1}$. If $x^*_{F,i} < L_{a+1}$, then we set $L_{a+1} = L_{a+1} - x^*_{F,i}$, $x^*_{F,i} = 0$ and consider the next partition $P_{F,i+1}$. Otherwise we reduce the length $x^*_{F,i}$ by $L_{a+1}$ and go to the next interval $[t^*_{a+1}, t^*_{a+2})$.

Since $\sum_{F\in\mathcal{F}}\sum_{i=1}^{n_F} a_\ell(F,i)\cdot x^*_{F,i} \;\geq\; \hat{D}^\ell$, this procedure completely schedules all $\ell$-processor tasks, and it runs in $O(n)$ time. However it is possible that some tasks are preempted, that some $\ell$-processor tasks are assigned to several blocks of $\ell$ processors or that some tasks can get different processor assignments in different intervals. The following lemma gives an upper bound on the number of incorrectly scheduled tasks.

Lemma 3.5. $|\mathcal{U}| \;\leq\; 2(m-1)k + 2m^{m+1}$.

*Proof.* First, we may assume that all partitions with $m$ free processors are in the last interval $[t_{g-1}, t_g)$; otherwise we can shift all such intervals at the end of the schedule. There are two different cases of incorrectly scheduled tasks: 1. inside of an interval $[t_i, t_{i+1})$ caused by a change from a partition $P_{F,a}$ to $P_{F,a+1}$; 2. at the end of an interval $[t_i, t_{i+1})$ or inside of an interval without a change of partitions. In the second case, we have at most $(m-1)$ new incorrectly scheduled tasks (not counted before) for each interval $[t_i, t_{i+1})$. In the interval $[t_{g-1}, t_g)$, the last selected task is either not preempted or counted before. In total, we obtain at most $(m-1)g \;\leq\; 2(m-1)k$ incorrectly scheduled tasks in case 2. In the first case, for each change from a partition $P_{F,a}$ to $P_{F,a+1}$ we get at most $m$ incorrectly scheduled tasks. The number of all partitions can be bounded by $m! + m^m \leq 2m^m$, which implies the bound $2m^{m+1}$ for incorrectly scheduled tasks in case 1. $\square$

The following bound is a corollary of those in Lemmas 3.4 and 3.5.

Corollary 3.1. $|\mathcal{V}\cup\mathcal{U}| \;\leq\; (2m-1)k + 3m^{m+1}$.

## 4 Analysis of the Algorithm

The following lemmas generalize some of the results in [1]. The first one provides a bound on the number of relative schedules in terms of $k$ and $m$, and the second one shows how the constant $k = k(m,\epsilon)$ has to be selected.

Lemma 4.1. *If $|\mathcal{L}| = k > 0$, then the total number of relative schedules of $\mathcal{L}$ is at most $(2^{m+2}k^2)^k$.*

*Proof.* Since there are only $2^m - 1$ different non-empty subsets of $\{1,\ldots,m\}$, there are at most $(2^m-1)^k$ possible processor assignments for the tasks in $\mathcal{L}$. A snapshot is created when a task of $\mathcal{L}$ starts or ends its execution. Since $|\mathcal{L}| = k$, there are at most $2k$ such events, and consequently, at most $2k$ snapshots. Furthermore, for each task $T_j \in \mathcal{L}$, the first and last snapshots containing $T_j$ have to be chosen. There are $4k^2$ possible choices for each task, and therefore the number of different sequences of snapshots is bounded by $(4k^2)^k$. Thus in total there are at most $(2^{m+2}k^2)^k$ different relative schedules. $\square$

Lemma 4.2. *Suppose* $d_0 \geq d_1 \geq \ldots \geq d_{n-1} > 0$ *is a sequence of real numbers and* $D = \sum_{i=0}^{n-1} d_i$. *Let* $m$ *be a positive integer,* $\mu > 0$, *and assume that* $n \geq 4m^{m+1}(2m)^{\lceil \frac{1}{\mu} \rceil}$. *Then there exists an integer* $k \leq 4m^{m+1}(2m)^{\lceil \frac{1}{\mu} \rceil - 1}$ *such that*

$$d_k + d_{k+1} + \ldots + d_{2mk+3m^{m+1}-1} \leq \mu \cdot D.$$

*Proof.* Decompose the sum $d_0 + \ldots + d_{n-1}$ into blocks $B_0 = d_0 + \ldots + d_{f(1)-1}$, $B_1 = d_{f(1)} + \ldots + d_{f(2)-1}$, $\ldots$, $B_i = d_{f(i)} + \ldots + d_{f(i+1)-1}$, where the function $f(i)$ is defined recursively by the following equations

$$f(0) = 0,\ f(1) = 2m + 3m^{m+1},\ f(i+1) = 2m \cdot f(i) + 3m^{m+1}, \quad i = 1, 2, \ldots \qquad (4.3)$$

Since $\sum_{j=0}^{n-1} d_j = D$, at most $\lceil \frac{1}{\mu} \rceil - 1$ blocks have size larger than $\mu \cdot D$. Now let $i$ be the smallest integer for which $B_i \leq \mu \cdot D$. Then $i \leq \lceil \frac{1}{\mu} \rceil - 1$, and $B_i = d_{f(i)} + \ldots + d_{f(i+1)-1} \leq \mu \cdot D$. This implies that there is an index $k \leq f(i)$ such that $d_k + \ldots + d_{2mk+3m^{m+1}-1} \leq \mu \cdot D$. It follows from (4.3) that

$$f(i) = (2m)^i \ + \ 3m^{m+1}(1 + 2m + \ldots + (2m)^{i-1}),$$

and thus

$$f(i) \ \leq \ \frac{(3m^{m+1} + 2m - 1)(2m)^i - 3m^{m+1}}{2m-1} \ \leq \ 4m^{m+1}(2m)^i,$$

which along with the bound on $i$ implies the lemma. □

By considering all relative schedules for $\mathcal{L}$, Lemma 3.3 can guarantee that the makespan for the partial (feasible) schedule of $\mathcal{T} \setminus (\mathcal{V} \cup \mathcal{U})$ is bounded by $OPT + \frac{\epsilon}{2} \cdot OPT$. Corollary 3.1 and Lemma 4.2 with $\mu = \frac{\epsilon}{2m}$ imply that the tasks in $\mathcal{V} \cup \mathcal{U}$ can be scheduled (at the end) with a total length of at most $\frac{\epsilon}{2} \cdot OPT$. Thus the overall makespan of the (complete) schedule is bounded by $(1 + \epsilon) \cdot OPT$. According to the arguments above, for every fixed $m$ and $\epsilon$, all computations can be carried out in $O(n)$ time. Thus the following result has been proved.

Theorem 4.1. *For any fixed* $m$, *there is a polynomial-time approximation scheme for the MPTS problem that computes for any fixed* $\epsilon > 0$ *in* $O(n)$ *time a feasible schedule whose makespan is at most* $(1 + \epsilon)$ *times the optimum.*

For NPTS, we obtain a simpler and faster (linear time) approximation scheme, since in this case the $y_{j\ell}$ variables are not needed and so the the corresponding linear program has only a constant$(m, \epsilon)$ number of variables and constraints.

## 5 Instances with Work Constraints

In this section we consider MPTS under the following (natural) assumption: For each task $T_j \in \mathcal{T}$ and each number $p \in M$, the work/execution time $t_j(1)$ on 1 processor is bounded by the work $pt_j(p)$ on $p$ processors. This assumption is motivated by the fact that in general, executing a task on $p$ processors requires communications between the processors, and therefore the overall work on $p$ processors is at least as large as on 1 processor. The time for communications among processors that work on the same task can be taken into account implicitly in the execution time. In the following we show that under this assumption the previous linear programs and therefore also the approximation schemes substantially simplify.

The idea is to transform a short task $T_j$ running on $p$ processors into $p$ parallel sequential tasks $T_j^1, \ldots, T_j^p$, where each of these $p$ tasks has the same execution time of at most $t_j(p)$ (see Figure 3).

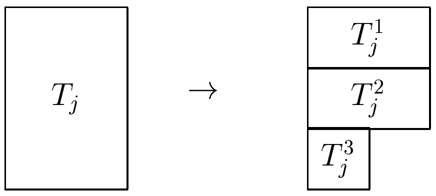

Figure 3: Transformation of a 3-processor task $T_j$ into 3 parallel 1-processor tasks $T_j^1, T_j^2, T_j^3$.

In general the produced schedules are not feasible, because the 1-processor short tasks are executed now in parallel. But by using the approach of Section 3.4, one can put the short tasks into the gaps with free processors (in the relative schedule) such that the number of incorrectly scheduled tasks is bounded by a constant. The advantage of this new approach is that the $y_{j\ell}$ variables are not needed any longer. Therefore by assuming that every short task is executed on 1 processor, the linear program (for each relative schedule of the long tasks in $\mathcal{L}$) can be simplified as follows:

**Minimize** $t_g$ **s.t.**

**(0)-(2)**

**(3)'** $\sum_{i:|P(i)|=m-j} (t_i - t_{i-1}) = e_j, \quad j = 1, \ldots, m,$

**(4)'** $\sum_{j=1}^{m} j \cdot e_j \geq D^1.$

In this linear program, the variable $e_j$ corresponds to the processing time while $m-j$ processors are executing long tasks of $\mathcal{L}$ and $j$ processors are free for short tasks. The value $D^1$ gives the total processing time $\sum_{T_j \in \mathcal{T} \setminus \mathcal{L}} t_j(1)$ of all short tasks on 1 processor. Inequality $(4)'$ guarantees that there is enough time for all short tasks. By using a similar argument as before the number of incorrectly scheduled tasks can be bounded in terms of $m$ and $\epsilon$, and therefore one can execute them sequentially at the end of the schedule with a total execution time of at most $\frac{\epsilon}{2} \cdot OPT$. Since now the linear programs have only a constant$(m, \epsilon)$ number of variables and constraints, one can get a faster approximation scheme for this case.

## 6 Conclusions

We mention in closing that by using similar ideas linear time approximation schemes can also be obtained for the following scheduling problems with makespan minimization on a fixed number of machines: preemptive versions of NPTS and MPTS; preemptive and non-preemptive scheduling of unrelated parallel machines (with and without cost); flow and open shop scheduling, $\mathrm{P}m|\mathrm{set}_j|\mathrm{C}_{\max}$. In fact, these linear-time approximation schemes, except for the last three models, are fully polynomial. We plan to address these problems along with other extensions of the above results in a subsequent paper.

This work was motivated by [1], whose authors (by extending their previous results) have also obtained independently a linear-time approximation scheme for NPTS with a fixed number of processors [2].

## References

[1] A. K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis, *Scheduling independent multiprocessor tasks*, Proc. 5th European Symposium on Algorithms (1997), LNCS 1284, pp. 1–12.

[2] A.K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis, private communication.

[3] K. Belkhale and P. Banerjee, *Approximate algorithms for the partitionable independent task scheduling problem*, International Conference on Parallel Processing (1990), Vol. 1, pp. 72–75.

[4] B. Baker, E. Coffman, and R. Rivest, *Orthogonal packings in two dimensions*, SIAM Journal on Computing 9 (1980), pp. 846–855.

[5] J. Blazewicz, W. Cellary, R. Slowinski, and J. Weglarz, *Scheduling under resource constraints - deterministic models*, Annals of Operations Research 7 (1986).

[6] E. Coffman, M. Garey, D. S. Johnson, and R. Tarjan, *Performance bounds for level-oriented two-dimensional packing problems*, SIAM Journal on Computing, 9 (1980), pp. 808–826.

[7] J. Du and J. Leung, *Complexity of scheduling parallel task systems*, SIAM Journal on Discrete Mathematics, 2 (1989), pp. 473–487.

[8] A. Feldmann, M.-Y. Kao, J. Sgall, and S. H. Teng, *Optimal online scheduling of parallel jobs with dependencies*, Proc. 25th ACM Symposium on the Theory of Computing (1993), pp. 642–651.

[9] A. Feldmann, J. Sgall, and S.H. Teng, *Dynamic scheduling on parallel machines*, Proc. 32nd IEEE Symposium on Foundations of Computer Science (1991), pp. 111–120.

[10] M. Garey and R. Graham, *Bounds for multiprocessor scheduling with resource constraints*, SIAM Journal on Computing 4 (1975), pp. 187–200.

[11] M.D. Grigoriadis and L.G. Khachiyan, *Coordination complexity of parallel price-directive decomposition*, Mathematics of Operations Research 21 (1996), pp. 321–340.

[12] M. Garey and D.S. Johnson, *Complexity results for multiprocessor scheduling under resource constraints*, SIAM Journal on Computing 4 (1975), pp. 397–411.

[13] C. Kenyon and E. Remila, *Approximate strip packing*, Proc. 37th IEEE Symposium on Foundations of Computer Science (1996), pp. 31–36.

[14] Y. Kopidakis and V. Zissimopoulos, An approximation scheme for scheduling independent jobs into subcubes of a hypercube of fixed dimension, *Theoretical Computer Science* (1997), pp. 265–273.

[15] Y. Kopidakis, *Approximabilite des Problemes d 'Ordonnancement dans les Systemes Paralleles*, These de Doctorat, Universite de Paris-Sud (1997).

[16] W. Ludwig and P. Tiwari, *Scheduling malleable and nonmalleable parallel tasks*, Proc. 5th ACM-SIAM Symposium on Discrete Algorithms (1994), pp. 167–176.

[17] S.A. Plotkin, D.B. Shmoys, and E. Tardos, *Fast approximation algorithms for fractional packing and covering problems*, Mathematics of Operations Research 20 (1995), pp. 257–301.

[18] A. Steinberg, *A strip packing algorithm with absolute performance bound 2*, SIAM Journal on Computing 26 (1997), pp. 401–409.

[19] J. Turek, J. Wolf, and P. Yu, *Approximate algorithms for scheduling parallelizable tasks*, Proc. 4th ACM Symposium on Parallel Algorithms and Architectures (1992), pp. 323–332.