Efficient Approximation Algorithms for Scheduling Malleable Tasks

Grégory Mounié, Christophe Rapine, Denis Trystram

Efficient Approximation Algorithms for Scheduling Malleable Tasks

Gregory Mounie, Christophe Rapine and Denis Trystram
IMAG, Domaine Universitaire BP 53
38041 Grenoble cedex, France
{Gregory.Mounie, Christophe.Rapine, Denis.Trystram}@imag.fr

Abstract

A *malleable task* is a computational unit which may be executed on any arbitrary number of processors, its execution time depending on the amount of resources allotted to it. According to the standard behavior of parallel applications, we assume that the malleable tasks are monotonic, *i.e.* that the execution time is decreasing with the number of processors while the computational work increases. This paper presents a new approach for scheduling a set of independent malleable tasks which leads to a worst case guarantee of $\sqrt{3}$ for the minimization of the parallel execution time, or *makespan*. It improves all other existing practical results including the two-phases method introduced by Turek et al. The main idea is to transfer the difficulty of a two phases method from the scheduling part to the allotment selection. We show how to formulate this last problem as a knapsack optimization problem. Then, the scheduling problem is solved by a dual-approximation which leads to a simple structure of two consecutive shelves.

1 Introduction

Until recently, the standard communication model for scheduling the tasks of a parallel program has been the *delay* model introduced by Rayward-Smith [16] for UET-UCT task graphs (unit execution times and unit communication times). In this model, the communications between tasks allocated to different processors are considered explicitly by the transmission time of a message between them. The communication times between tasks within the same processor are neglected. The general scheduling problem is known to be NP-hard in the strong sense [16]. However, efficient heuristics have been developed for coarse grain computations, i.e. when communication times are smaller than computation times. Unfortunately, this hypothesis is not often valid in practice. Only a few results have been obtained under the delay model for large communication times. A good alternative to take into account the communications is to consider the *malleable tasks* model where the communication times are considered implicitly by a function representing the parallel execution time with the penalty due to the management of the parallelism. As detailed later, this function will be considered as *monotonic*, i.e. decreasing with the number of processors, at least until a certain threshold. More formally, a malleable task is a computational unit which may be executed in parallel with a running time depending on the number of processors allotted to it. Some authors recently used this model for parallelizing actual applications. We are currently developing such a code for the simulation of the circulations in the Atlantic Ocean [3]. In this paper, we are interested in scheduling a set of n independent malleable tasks on a multiprocessor system composed by m identical processors. Our main contribution is to propose a new efficient heuristic for the case of independent malleable tasks with a performance guarantee of $\sqrt{3}$. This bound improves all existing practical results for solving this problem.

Several methods have been proposed for the scheduling of malleable tasks. Considering the processors as a continuous resource, Prasanna et al. [14, 15] developed an approach based on optimal control theory leading to optimal solution for general task graphs but with an identical particular speedup function for the tasks. However, the discrete problem remains NP-hard, even for independent tasks, as a generalization of the classical multiprocessors scheduling problem [7]. Jansen & Porkolab [10] proposed a fully approximable scheme based on an Integer Linear Programming formulation for scheduling independent malleable tasks. Although linear in the number of tasks the complexity of the scheme is quite large, regardless the accuracy of the approximation, due to an exponential factor in the number of processors. Thus, even if the result has an important theoretical interest, the heuristic can not really be used in practice.

We are interested in efficient, low complexity, heuristics with good performance guarantee. Most existing works are based on a two-phases approach proposed by Turek, Wolf and Yu [18]. The basic idea is to select in a first step an allotment (the number of processors allotted to each task), and then solve the resulting non-malleable scheduling problem. As far as the makespan criterion is concerned, this latter problem is identical to a 2-dimensional strip-packing problem [2, 5]. It is clear that applying an approximation of guarantee λ for the non-malleable problem on the allotment of an optimal solution provides the same guarantee λ for the malleable problem. Turek, Wolf and Yu established the elegant result that the allotment selection can be done in polynomial time: any λ-approximation algorithm of complexity $\mathcal{O}(f(n,m))$ for the non-malleable problem can be adapted into a λ-approximation algorithm of complexity $\mathcal{O}(mnf(n,m))$ for the malleable problem.

Ludwig [12] improved the complexity of the allotment selection. Based on this result and on the 2-dimensional strip-packing algorithm of guarantee 2 proposed [1] for 2-dimensional strip-packing by Steinberg [17], he presented an heuristic of guarantee 2 for the malleable scheduling problem. This is the best efficient solution known at this time. Note that this solution considers any speedup function. Again, our solution needs the monotonic assumption which is quite reasonable in Parallel Processing.

We also adopt a two phases approach, but contrary to the method of Turek et al. where the complexity basically comes from the scheduling phase, which is a $\mathcal{NP}$-hard problem, we concentrate on the first phase, the allotment selection, polynomial in Turek's approach. More precisely, we propose to select an allotment such that we do not have to solve a general strip-packing instance, but a simpler one where better performance guarantees can be ensured. Our second idea is that scheduling (or packing) tasks executed on one processor is "easier" than scheduling parallel tasks, i.e. better guarantees can be obtained. Based on this two ideas we propose mainly two approaches depending on an instance parameter S_m. If this parameter happens to be small, we show in section 3 that list scheduling algorithms perform quite well. In fact our allotment selection ensures a relatively small number of parallel tasks to schedule, while sequential tasks are easily managed. In the case of a large instance parameter S_m, we decide to fix the structure of the scheduling we want to obtain. The allotment selection is then reformulated as a Knapsack optimization problem.

The paper is organized as follows: some basic assumptions and simple notations are presented in section 2. In particular, the monotonic assumptions will be presented and discussed. Section 3 aims to describe the list scheduling approach and its performance analysis. Finally, we present the most original part of the heuristics, based on a Knapsack approximation for the allotment selection.

2 Preliminaries

The model of computation is such that a processor can compute only one task at a time. We assume also that the number of processors allocated to a task is constant during all its execution. We search for non-preemptive and contiguous[2] schedules; their performance guarantee are established in respect to an optimal solution, possibly preemptive and not contiguous. We consider an instance composed of n malleable tasks $\{T_1, \ldots, T_n\}$ to be scheduled on m identical processors. The execution time of the malleable task T_i when allotted to p processors will be denoted by $t_{i,p}$. Its *computational area* is as usual the time space product $pt_{i,p}$.

2.1 Monotonic assumptions

In this paper, we assume that the tasks are monotonic, namely allocating more processors to a task decreases its execution time and increases its computational area. From the parallel computing point of view, this monotonic assumptions may be interpreted by the well-known Brent's lemma [4]. It states that the execution of malleable tasks achieves some speedups, but not super-linear speedups. Due to cache effects or the scheduling anomalies described by Graham [8], this behavior can not be asserted for all the applications. However, it is a quite reasonable hypothesis, expected for most actual parallel applications, mainly due to the communication overhead. In the following we will use the notion of dual approximation algorithm described in section 2.2. Let d be a real number. Since we will "guess" the value of the optimal makespan, we introduce the minimal number of processors (denoted p_i^d) for executing task T_i in time less than d. We now state two fundamental properties which are two direct consequences of the monotonic hypothesis.

Property 1 *If p_i^d exists, then $t_{i,p_i^d} > \frac{p_i^d - 1}{p_i^d} d$.*

This property is immediate simply writing the decreasing of the computational area. Indeed we have $p_i^d t_{i,p_i^d} \geq (p_i^d - 1) t_{i,p_i^d - 1}$. Notice simply that by definition of p_i^d we have $t_{i,p_i^d - 1} > d$.

Property 2 *Assume that a schedule of length lower than d exists. Then for any allotment q_i such that $q_i \leq p_i^d$ for any i, we have $\sum_{i=1}^{n} q_i t_{i,q_i} \leq md$*

If a schedule of length lower than d exists, clearly any task T_i is allocated to at least p_i^d processors. Due to the monotony of the computational area, the allotment using the q_i's has a total area lower than the one of the p_i's. However this last area is bounded by md, since the makespan of the schedule does not exceed d.

2.2 Dual approximation algorithm

We use in this paper the dual approximation methods introduced by Hochbaum & Shmoys [9]. Given a real number d, a dual λ-approximation:

- either delivers a schedule of length at most λd,
- or conclude that no schedule of length lower than d exists.

By dichotomic search, a λ-dual approximation of time complexity $\mathcal{O}(f(n))$ can be converted into a $\lambda(1 + 2^{-k})$-approximation of complexity $\mathcal{O}(kf(n))$ for any integer k.

3 List Scheduling Algorithms

In a two phases approach for the malleable scheduling (MS) problem, we have both to select an allotment and to solve the resulting non malleable scheduling problem (NMS) problem where the number of processors allocated to any task is fixed. As far as the makespan criterion is considered, the NMS problem is equivalent to a 2-dimensional strip-packing problem. Hence, to improve the guarantee of 2 of Ludwig [12], it seems necessary to improve the absolute performance guarantee of the strip-packing. However, under the monotonic hypothesis, we show that the allotment selection can be done in such a way we do not have to solve a general instance of the strip-packing problem, but a simpler one, for which list scheduling algorithms perform quite well. Graham & Johnson [6] proved a general performance guarantee of $2s$ for any list scheduling algorithm for a set of independent tasks sharing s different resources. It leads to a worst case performance of 2 for the NMS problem, since the processors are the only (discrete) resource of the system. To improve this guarantee, we propose below a simple dual algorithm of guarantee $2 - 2/(m+1)$.

3.1 Malleable List Algorithm.

Without loss of generality assume that a schedule of length lower than 1 exists for the malleable instance. Then the following two phases malleable list algorithm builds up a schedule of length at most $2 - 2/(m+1)$. Fig. 1 depicts the topical structure.

- Allotment: Select for any task T_i its minimal number of processors p_i to assert an execution time lower than $2 - \frac{2}{(m+1)}$.
- Scheduling: Apply a list algorithm to the resulting non-malleable instance ordered by decreasing value of the *sequential* execution times.

[1] Although better *asymptotic* performance guarantees exist [1], this bound of 2 constitutes the best currently known *absolute* performance guarantee.

[2] The processors alloted to a task have consecutive indices, which limits the communication overhead inside a task.

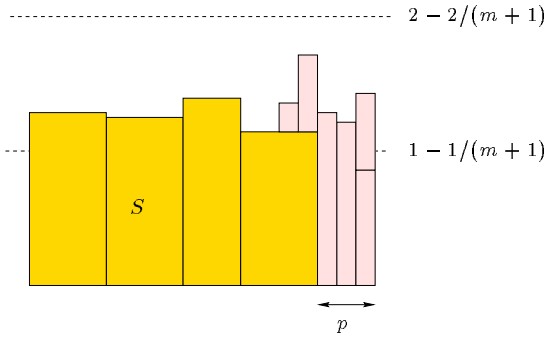

Figure 1: *A malleable list schedule. The parallel tasks are represented in dark grey.*

Theorem 1 *The Malleable List Algorithm is a* $2 - \frac{2}{m+1}$ *dual approximation for the MS problem.*

Proof. The proof is based on the property that the *parallel tasks*, i.e. the tasks alloted to 2 or more processors, are all executed in parallel at time 0 in the schedule. This property meets our philosophy of avoiding a general strip-packing instance, as the remaining tasks of the instance constitute a classical multiprocessor scheduling problem of sequential tasks. Let S be the total area of the parallel tasks of the allotment, and m' the sum of their number of processors. Notice that these tasks are scheduled first in the algorithm. Due to property 1 any parallel task has an execution time strictly greater than $1 - \frac{1}{m+1}$. Thus $S > (1 - \frac{1}{m+1})m'$. Moreover property 2 implies that $m \geq S$. It results that $m' < m + 1$. Thus the set of the parallel tasks needs at most the whole resources for their execution. We denote by $p = m - m'$ the remaining number of unoccupied processors after their allocation. Only sequential tasks remain to schedule, the scheduling algorithm is identical to the well-known *LPT* [8] heuristic. Consider the allocation of a sequential task T of duration t. We establish that T finishes before time $2 - \frac{2}{m+1}$. To avoid trivial cases, we can assume that at least p sequential tasks have yet been allocated. We write the area S under the form $(m - p) - s$, with $s \leq (m - p)/(m + 1)$. Then the total area of the sequential tasks is bounded by $p + s$. Let h be the starting time of T. We write then the conservation of the sequential task area, using the *LPT* rules. We both have $ph + t \leq p + s$ (all p processors are fully occupied until time h) and $(p+1)t \leq p+s$ (at least p sequential tasks have been allocated before T). It implies that:

$$h + t \leq \frac{p+s}{p} + (1 - \frac{1}{p})t \leq \frac{p+s}{p}\frac{2p}{p+1}$$

Since $s \leq (m - p)/(m + 1)$, we obtain:

$$h + t \leq \frac{m(p+1)}{p(m+1)}\frac{2p}{p+1} \leq \frac{2m}{m+1}$$

which proves that T finishes before time $2 - \frac{2}{m+1}$. □
Notice that if the number of processors is at most 6, the algorithm has a performance guarantee lower than $\sqrt{3}$. To extent this result, we detail in the next section another list scheduling algorithm.

3.2 The canonical list algorithm

Consider an instance of the MS problem and assume that it admits a schedule of length at most d, we take without loss of generality equal to 1. Our goal is to construct a 2λ dual approximation for a given real number $\lambda \in]3/4, 1]$. Compared to the previous malleable list algorithm, a more natural approach is to select for any task the minimal number of processors, (called the *canonical* number) such that its execution does not exceed 1. Notice that in the optimal scheduling each task is executed on at least its canonical number of processors. In addition, the decreasing actual execution time of the tasks is a more natural ordering of the list than their sequential length. We consider the following dual algorithm:

Canonical List Algorithm. Assume that a schedule of length lower than 1 exists for the considered malleable instance.

- Allotment: Select for any task T_i its *canonical* number of processors p_i defined as the minimal number of processors to assert an execution time lower than 1.
- Scheduling: Apply a list scheduling algorithm to the resulting non-malleable instance ordered by decreasing value of the execution times t_{i,p_i}.

In case of tie between the processors, the task is scheduled to the leftmost ones, if starting at time 0, and to the rightmost otherwise. This convention asserts the contiguous nature of the schedule. To get a guarantee 2λ for any real number $\lambda \in]3/4, 1]$ we will need an additional hypothesis. By the way to avoid the complexity of a general strip-packing problem, the scheduling structure of the parallel tasks, i.e. the tasks allocated to 2 or more processors in the allotment, must remain simple. Typically we hope for a 2-shelves like structure for these tasks. More precisely we call *first level* the set of tasks allocated at time 0; the *second level* corresponds to the tasks scheduled on top of a task of the first level, see fig 2. We would like to get the following property, for some $\lambda \in]3/4, 1]$:

Property 3 *In the schedule any task of the first two levels finishes before time* 2λ.

We show in appendix that there exists a constant m_λ depending only of λ such that for any number of processors $m \geq m_\lambda$ the property 3 is verified by the scheduling. For $\lambda = \sqrt{3}/2$, this minimal number of processors is precisely 7, i.e. the first value of m for which the guarantee of the previous malleable list algorithm becomes greater than $\sqrt{3}$. However to assert this property a given real λ must satisfy the additional hypothesis $S_m \leq \lambda m$. S_m represents a peculiar area depending only on the canonical allotment of the instance. To define the quantity S_m assume that the tasks are indexed by decreasing value of their canonical execution time t_{i,p_i}. We adopt the following definition:

Definition 1 *If* k *is the minimal index such that* $\sum_{i=1}^{k} p_i \geq m$*, we define the canonical* m*-area* S_m *as:*

$$S_m = \sum_{i=1}^{k} p_i(t_{i,p_i} - t_{k,p_k}) + mt_{k,p_k}$$

To visualize this area, imagine an execution of the canonical list algorithm on an unbounded number of processors. S_m then corresponds to the (fractional) area computed by the first m processors. Since under the hypothesis on S_m the property 3 holds, we can focus on the tasks scheduled outside the first two levels. The lemma 1 shows that these tasks are "small" sequential task, and complete before $3/2$:

Lemma 1 *Any task that is not scheduled in the first two levels is sequential, with an execution time lower than* $1/2$*, and finishes before time* $3/2$.

Proof. The proof of the sequential nature of the task are quite simple arguments of work conservation, and have been put in appendix. We present here the proof that such a sequential task T, of execution time $t \leq 1/2$, completes before time $3/2$. To be able to use an argument on the surface of the schedule, we need to bound quite precisely its different idle periods. Since outside the first two levels tasks are sequential and scheduled as soon as possible, idle time can appear only between these two levels, like between two stairs badly adjusted, as depicted in figure 2. Consider an connected idle

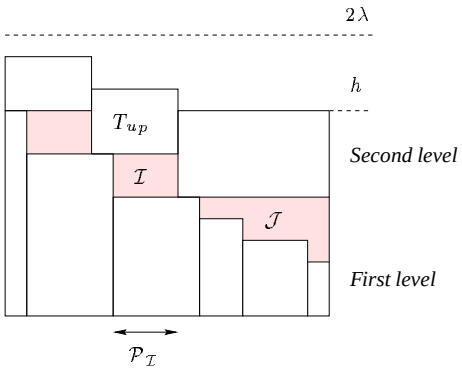

Figure 2: The idle areas in the schedule

area $\mathcal{I}$ occuring on a set of $p_{\mathcal{I}}$ contiguous processors $\mathcal{P}_{\mathcal{I}}$. The important point is that only one task T_{up} can upper delimit this idle area, see fig 2. This task belongs to the second level and is executed at least on $p_{\mathcal{I}} + 1$ processors, involving (property 1) an execution time t_{up} greater than $p_{\mathcal{I}}/(p_{\mathcal{I}} + 1) > 1/2$. Let denote by $S_{\mathcal{I}}$ the total area computed by the processors of $\mathcal{P}_{\mathcal{I}}$. We particularize the idle area $\mathcal{J}$ that may happen on the last processors. Indeed if $\mathcal{I}$ is different from $\mathcal{J}$, the processors of $\mathcal{P}_{\mathcal{I}}$ compute on the first level tasks of height at least t_{up}. It involves that $S_{\mathcal{I}} \geq 2t_{up}p_{\mathcal{I}} \geq p_{\mathcal{I}}$.

For the last idle area $\mathcal{J}$, we can only assert that any processor of $\mathcal{P}_{\mathcal{J}}$ compute on the first level tasks of height at least t, the length of the current task to schedule. Hence we get $S_{\mathcal{J}} \geq (t_{up} + t)p_{\mathcal{J}}$. Nevertheless task T_{up} can not complete before time $2t_{up}$ since it lays on a task of the first level greater than it. It results that $S_{\mathcal{J}}$ is also greater than $2t_{up}p_{\mathcal{J}} - \mathcal{J}$. Noticing that the area of $\mathcal{J}$ is lower than $p_{\mathcal{J}}t$, otherwise T could be executed inside this area, we get the following bound:

$$S_{\mathcal{J}} \geq \max\{2t_{up} - t, t_{up} + t\}p_{\mathcal{J}} \geq p_{\mathcal{J}} - t$$

Consider now the total area of the scheduling, and let $\mathcal{I}_1, \ldots, \mathcal{I}_l$, $\mathcal{I}_{l+1} = \mathcal{J}$ be the different idle areas. Clearly the sets $\mathcal{P}_{\mathcal{I}_i}$ are two by two disjoint and their union is a strict subset of the processors. The task T being sequential, it is allocated to the less loaded processor. Let h be its starting time in the schedule; by definition any processor computes at least an area h in the schedule. Since we established that the execution time t of T is lower than $1/2$, to prove the lemma we just need to show that necessarily $h \leq 1$. Writing down the area of the schedule we have:

$$m \geq \sum_{i=1}^{l} S_{\mathcal{I}_i} + S_{\mathcal{J}} + (m - \sum_{i=1}^{l} p_{\mathcal{I}_i} - p_{\mathcal{J}})h + t$$

Using $S_{\mathcal{J}} + t \geq p_{\mathcal{J}}$ and $S_{\mathcal{I}_i} \geq p_{\mathcal{I}_i}$, we get:

$$m - \sum_{i=1}^{l} p_{\mathcal{I}_i} \geq (m - \sum_{i=1}^{l+1} p_{\mathcal{I}_i})h$$

Since not all processors can have an idle period, the quantity $m - \sum_{i=1}^{l} p_{\mathcal{I}_i}$ is strictly positive, which implies that h is lower than 1. It involves that T completes in the schedule before time $3/2$. □

From property 3 and lemma 1, we can summarize the results of this part with the following theorem:

Theorem 2 *For any instance admitting an optimal schedule with makespan lower than* 1 *on* $m \geq 7$ *processors, such that the canonical* m*-area* S_m *is lower than* $\sqrt{3}/2m$*, the canonical list algorithm delivers a schedule of length at most* $\sqrt{3}$*, in time complexity* $\mathcal{O}(n \log m)$.

4 A new formulation of the tasks allotment problem

The idea of this section is to focus on the allotment selection problem rather than on the scheduling part. Consider an instance of the MS problem for which a schedule of length at most 1 exists on m processors. We consider a real number $\lambda \in]\frac{1}{2}, 1]$. If the area S_m defined in the canonical allotment is small enough, i.e. $S_m \leq \frac{1+\lambda}{2}m$, then for a value of m sufficiently large, the list algorithm of the previous section provides a schedule of makespan lower than $1+\lambda$. We propose in this section a new scheduling algorithm in the case where the condition on S_m is not fulfilled, which also delivers a solution of length at most $1 + \lambda$. Our basic idea is to impose the solution to have a very simple scheduling structure, namely composed by two shelves of length 1 and λ. In other words, instead of looking for one general scheduling, i.e. solving a strip-packing problem, we will split the instance in two subsets of tasks, each one admitting a "trivial" scheduling of length lower than 1 and λ respectively. The complexity of the two phases algorithms is then taken back from the scheduling phases to the choice of such a partition of the tasks. We will show in the following that this choice can be naturally formulated as a Knapsack optimization problem. First, we will precisely describe the structure of the scheduling we want to obtain.

4.1 The scheduling structure

Recall that p_i denotes the canonical number of processors of task T_i, defined as the minimal amount of resources needed by T_i to be executed in time lower than 1. Basically we want a two shelves structure, with a *first* shelf of length at most 1 and a *second* shelf of length at most λ, cf fig. 4. The tasks whose canonical execution time is greater than λ play, here, a particular role, since they require strictly more processors than their canonical number to be allocated in the second shelf. To obtain a reasonable scheduling structure we also particularize the "small" tasks with a canonical execution time lower than $1/2$, by allocating them to the shelves using a one dimensional packing algorithm, for instance the *First Fit* algorithm [11]. Notice that due to property 1, these small tasks are sequential, and thus using a one-dimensional packing algorithm makes sense. Hence we partition the tasks into 3 subsets $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$ (cf fig. 3), each one associated with an integer value π_i as defined below:

$$\begin{array}{ll} \mathcal{S}_1 = \{T_i \mid t_{i,p_i} > \lambda\} & \text{with } \pi_1 = \sum_{\mathcal{S}_1} p_i - m \\ \mathcal{S}_2 = \{T_i \mid t_{i,p_i} \in [\frac{1}{2}, \lambda]\} & \text{with } \pi_2 = \sum_{\mathcal{S}_2} p_i \\ \mathcal{S}_3 = \{T_i \mid t_{i,p_i} < \frac{1}{2}\} & \text{with } \pi_3 = FF(1, \mathcal{S}_3) \end{array}$$

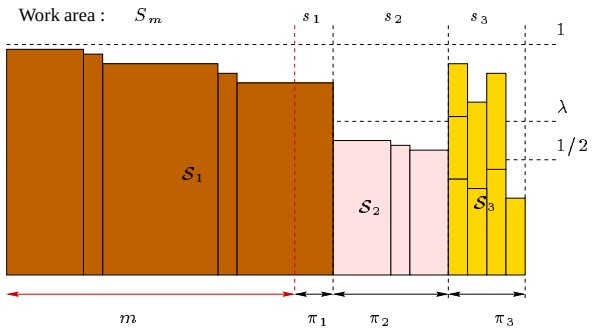

Figure 3: The initial canonical allocation

In this definition, $FF(d, \mathcal{S})$ refers to the number of processors needed by the *First Fit* algorithm to pack a set of tasks $\mathcal{S}$ under a time deadline d. Although better approximation algorithms are known for the one-dimensional packing problem, we just need the immediate property that if $FF(d, \mathcal{S}) > 1$ then the total amount of instructions $\sum_{\mathcal{S}} t_i$ of $\mathcal{S}$ is greater than $\frac{d}{2} FF(d, \mathcal{S})$.

With these notations we can build a schedule with a makespan lower than 1, but using $m + \pi_1 + \pi_2 + \pi_3$ processors, see figure 3. By definition the area of the schedule on the m first processors is equal to S_m. By analogy with the π's, we write respectively the canonical area of $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$ as $S_m + s_1$, s_2 and s_3. We consider now the following scheduling structure, we called a $2K$-schedule: for a task T_i, we denote by q_i its minimal number of processors needed for its execution to be lower than λ. Notice that if T_i does not belong to $\mathcal{S}_1$, we simply have $q_i = p_i$ by definition of $\mathcal{S}_2$ and $\mathcal{S}_3$. A schedule is then a $2K$-schedule if:

- The first shelf only contains tasks of $\mathcal{S}_1$ allotted to their canonical number of processors.
- Any task T_i of the second shelf is allocated to q_i processors; tasks of $\mathcal{S}_3$ are allocated to the shelf using the *First Fit* algorithm.

In other word we decide to consider the schedules where all the tasks of $\mathcal{S}_2$ and $\mathcal{S}_3$ are allocated in the second shelf. We denote by $\rho_2 = \pi_2$ the number of processors needed to schedule the tasks of $\mathcal{S}_2$, and by $\rho_3 = FF(\lambda, \mathcal{S}_3)$ the analogous quantity for $\mathcal{S}_3$, as depicted in figure 4. We set $\rho_1 = m - \rho_2 - \rho_3$ the number of idle remaining processors in the second shelf after $\mathcal{S}_2$ and $\mathcal{S}_3$ have been allocated. A $2K$-schedule is clearly entirely defined by the subset θ of $\mathcal{S}_1$ not scheduled in the first shelf. For the schedule to be feasible this subset θ must both have the sum of its p_i's greater than π_1, and the sum of its q_i's lower than ρ_1, which ensures that the other tasks of $\mathcal{S}_1$ can be scheduled in the first shelf, while the tasks of θ, $\mathcal{S}_2$ and $\mathcal{S}_3$ can be scheduled all together in the second shelf. We hence introduce the set $\Theta_\lambda \subseteq 2^{\mathcal{S}_1}$ defined by $\theta \in \Theta_\lambda$ iff:

$$\sum_{\theta} p_i \geq \pi_1 \quad \text{and} \quad \sum_{\theta} q_i \leq m - \rho_2 - \rho_3$$

Clearly there exists a bijection between the $2K$-schedules and the set Θ_λ. Our problem consists in proving that the set Θ_λ is not empty, i.e. that the problem instance admits a $2K$-schedule, and in finding one of its element. Before reformulating the search of an element of Θ_λ as a Knapsack problem, we introduce the notion of *inefficiency factor*, which permits us to keep a control on the optimal solution.

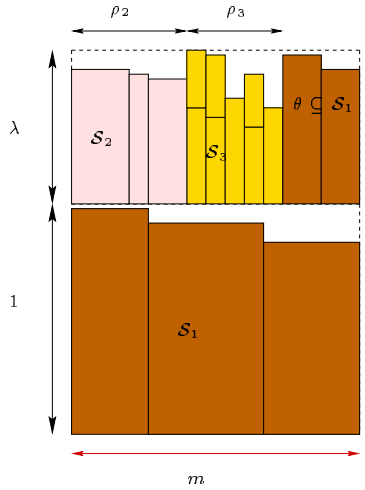

Figure 4: An example of $2K$-schedule

4.2 Inefficiency Factor

The inefficiency factor represents the factor of expansion of the total area of a set of tasks while using parallel resources. We take the canonical area of the tasks as the reference, since any task in the schedule, and in the optimal one, is allotted at least to its canonical number of processors. For a task T_i allocated to r processors, we define its inefficiency factor $\mu_{i,r}$ as $\mu_{i,r} = r t_{i,r} / (p_i t_{i,p_i})$. In the following we will denote for short by $a_i = p_i t_{i,p_i}$ the canonical area of task T_i. More generally for a set of tasks $\{T_1, \ldots, T_k\}$ allotted respectively to $r_1, \ldots, r_k$ processors, we set:

$$\mu_{\{T_1,\ldots,T_k\}} = \sum_{i}^{k} r_i t_{i,r_i} / \sum_{i}^{k} a_i = \sum_{i=1}^{k} \frac{\mu_{i,r_i} a_i}{a_1 + \ldots + a_k}$$

This inefficiency factor allows us to characterize our instance $\mathcal{S}$. Consider an optimal solution σ, whose makespan is hence by hypothesis lower than 1. In this schedule any task is alloted to at least its canonical number of processors to complete before time 1. It involves that not all the tasks of $\mathcal{S}_1$ can be scheduled on different processors if π_1 happens to be strictly positive, since they would need in this situation at least $m + \pi_1$ processors. Notice that for two tasks to be scheduled to the same processor, at least one of them must have an execution time lower than $1/2$ in σ.

Definition 2 *We denote by $\mathcal{A}$ the set of tasks of $\mathcal{S}_1$ executed in time less than $1/2$ in an optimal schedule of reference σ, and by $\mu_{\mathcal{A}}$ its inefficiency factor in σ.*

Our idea is to bound this quantity $\mu_{\mathcal{A}}$ according to λ and the canonical areas of the sets $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$. Recall that we have written respectively this three canonical areas as $S_m + s_1$, s_2 and s_3. By definition $S = S_m + s_1 + s_2 + s_3$. Due to the previous arguments, the sum of the canonical number of processors p_i on $\mathcal{A}$ must be greater than π_1. We denote by A and B respectively the canonical area of $\mathcal{A}$ and its actual area in the schedule. In addition let S and $\bar{A}$ be respectively the canonical area of the instance and of the complementary set of $\mathcal{A}$. By definition we have $A + \bar{A} = S$. The makespan of schedule σ being lower than 1, the monotony implies that $\bar{A} + B \leq m$. By definition of the inefficiency factor we also have: $\mu_{\mathcal{A}} = B/A \leq (m - S + A)/A$. This expression is

a decreasing function of A. However, for σ to be feasible, we must have $A \geq s_1$. Hence we get:

$$\mu_{\mathcal{A}} \leq \frac{m - S + s_1}{s_1} \leq \frac{m - S_m - s_2 - s_3}{s_1}$$

Since we assume that $S_m \geq (1+\lambda)/2m$, we can bound $\mu_{\mathcal{A}}$ by the following expression:

$$\mu_{\mathcal{A}} \leq \frac{(1-\lambda)m - 2s_2 - 2s_3}{2s_1}$$

4.3 Formulation as a Knapsack problem

Recall that our goal is to exhibit a $2K$-schedule for the malleable instance, or equivalently to show that the set Θ_λ is not empty and to find one of its element. Consider the following optimization problem:

$$(\mathcal{P}) : \text{Find } \mathcal{K} \subseteq \mathcal{S}_1 \quad \text{with } \sum_{\mathcal{K}} q_i \leq m - \rho_2 - \rho_3, \text{ maximizing } \sum_{\mathcal{K}} p_i.$$

Clearly if $\Theta_\lambda \neq \emptyset$, an optimal solution of $(\mathcal{P})$ belongs to it. The problem $(\mathcal{P})$ is a well-known Knapsack problem: given n items, each of one with a weight w_i and a profit v_i, and a knapsack with a total weight capacity W, find a subset of the tasks which can be contained by the knapsack and with the maximal profit. This problem is $\mathcal{NP}$-hard [7], hence solving it in an exact way seems to be unrealistic if $\mathcal{P} \neq \mathcal{NP}$. However the Knapsack is not a $\mathcal{NP}$-hard problem in the strong sense: it admits [13] a pseudo-polynomial algorithm that solves it exactly in time complexity $\mathcal{O}(nW)$. For our problem, m plays the part of the weight capacity W. Hence for "reasonable" values of the number of processors, an element of Θ_λ can be found solving exactly $(\mathcal{P})$ in time $\mathcal{O}(nm)$. Nevertheless if the different time values $(t_{i,r})_{r=1,m}$ are given by functions of size bounded in $\mathcal{O}(\log m)$, the instance of the malleable problem is of size $\mathcal{O}(n \log m)$, which makes the search of a solution of Θ_λ exponential if n happens to belong to $\mathcal{O}(\log m)$. Notice that the assumption $m \gg n$ does not "simplify" the malleable problem, contrary to the classical model where each task can be schedule only on one processor at a time. For this reason we propose a polynomial search algorithm of an element of Θ_λ whatever the size of the instance, based on the fully approximation scheme of the Knapsack problem rather than on its pseudo-polynomial algorithm.

4.4 Approximate solution of the Knapsack problem

To obtain a solution of problem $(\mathcal{P})$ in polynomial time, we use the fully approximable scheme [13] for the Knapsack problem: for any $\varepsilon > 0$, a solution within a factor $(1+\varepsilon)$ of the optimal weight value can be found in time complexity $\mathcal{O}(n^3/\varepsilon)$. Unfortunately what we are interested in is to find an element of Θ_λ, and we have no guarantee that an approximate solution of $(\mathcal{P})$ belongs to this set, contrary to the optimal solution. For this reason we introduce a second (dual) Knapsack problem:

$$(\mathcal{P}') : \text{Find } \mathcal{K} \subseteq \mathcal{S}_1 \quad \text{with } \sum_{\mathcal{K}} p_i \geq \pi_1, \text{ minimizing } \sum_{\mathcal{K}} q_i.$$

Our basic idea is that, for a real number ε_λ sufficiently small and depending only on λ, a $(1+\varepsilon_\lambda)$-approximation either of $(\mathcal{P})$ or $(\mathcal{P}')$ will deliver a solution belonging to Θ_λ. At this point we need to take advantage of the assumption that a schedule of length lower than 1 exists for the instance. This is done after introducing a subset Ξ_λ of Θ_λ which uses the inefficiency factor $\mu_{\mathcal{A}}$ of the optimal solution. Recall that a_i denotes the canonical area $p_i t_{ip_i}$ of task T_i; in the same way we denote b_i its work area when executed on q_i processors.

Definition 3 *We define Ξ_λ as the set of elements ξ of Θ_λ satisfying the additional condition:*

$$\sum_{\xi} b_i \leq \mu_{\mathcal{A}} \sum_{\xi} a_i$$

Basically the elements of Ξ_λ are reasonable guests to belong to the set $\mathcal{A}$ of the tasks of the optimal scheduling executed in time less than $1/2$. We now state the following lemma:

Lemma 2 *For all $\lambda \geq \sqrt{3} - 1$, if $\Xi_\lambda \neq \emptyset$, then a solution $\theta \in \Theta_\lambda$ can be found in time $\mathcal{O}(\frac{n^3}{\varepsilon_\lambda})$, where ε_λ is a strictly positive real number depending only on λ.*

Proof. Consider a real number $\varepsilon > 0$. We denote respectively by π^* and ρ'^* the optimal value of the objective functions of $(\mathcal{P})$ and $(\mathcal{P}')$. If we have $\pi^* \geq (1+\varepsilon)\pi_1$, clearly a $(1+\varepsilon)$-approximation of $(\mathcal{P})$ provides an element of Θ_λ. Otherwise assume that we have $\pi^* < (1+\varepsilon)\pi_1$. Our idea is to show that in this case we have $\rho'^* \leq \rho_1/(1+\varepsilon)$, which ensures that a $(1+\varepsilon)$-approximation of $(\mathcal{P}')$ delivers an element of Θ_λ. Since we suppose that Ξ_λ is not empty, we can choose one of its element ξ. Because of the optimality of π^* we have:

$$\pi_1 \leq \sum_{\xi} p_i \leq \pi^* < (1+\varepsilon)\pi_1$$

To get a majoration of ρ'^*, notice that for any subset of tasks of $\mathcal{S}_1$, and in particular for ξ, we have $\sum_\xi a_i \leq \sum_\xi p_i$ and $\frac{\lambda}{2}\sum_\xi q_i \leq \sum_\xi b_i$. Using the fact that $\xi \in \Xi_\lambda$, and hence $\sum_\xi b_i \leq \mu_{\mathcal{A}} \sum_\xi a_i$, we get:

$$\sum_{\xi} q_i < \mu_{\mathcal{A}} \frac{2}{\lambda} \sum_{\xi} p_i \leq \mu_{\mathcal{A}} \frac{2}{\lambda}(1+\varepsilon)\pi_1$$

Clearly ξ belongs to the space of solutions of $(\mathcal{P}')$. Reporting the majoration of $\mu_{\mathcal{A}}$ established in section 4.2, and using the fact that $\lambda\pi_1 \leq s_1$, we get:

$$\rho'^* < (1+\varepsilon)\frac{(1-\lambda)m - 2s_2 - 2s_3}{\lambda^2}$$

The function $f(\lambda) = (1-\lambda)/\lambda^2$ is decreasing and equals to 1 for $\lambda = \frac{\sqrt{5}-1}{2}$. For a sufficiently small constant $\varepsilon_\lambda > 0$, we can impose that $f(\lambda)(1+\varepsilon_\lambda)^2 \leq 1$. Hence for any $\varepsilon \leq \varepsilon_\lambda$ we have:

$$(1+\varepsilon)\rho'^* < m - \frac{2(1+\varepsilon_\lambda)^2}{\lambda^2}(s_2 + s_3)$$

To conclude, we explicit the relation between ρ_i and s_i. For the set $\mathcal{S}_2$ we clearly have $\frac{1}{2}\rho_2 < s_2 \leq \lambda\rho_2$. For the set $\mathcal{S}_3$ we also have the relation $\lambda/2\rho_3 < s_3 \leq \lambda\rho_3$ if $\rho_3 > 1$ due to the packing properties. Hence, we have $2s_2/\lambda^2 > \rho_2/\lambda^2 > \rho_2$ and $2s_3/\lambda^2 > \rho_3/\lambda > \rho_3$, which gives:

$$(1+\varepsilon)\rho'^* < m - \rho_2 - \rho_3 = \rho_1$$

which is the desired majoration. Notice that in case where $\rho_3 \leq 1$, choosing ε_λ such that $f(\lambda)(1+\varepsilon_\lambda)^2 \leq \frac{m-1}{m}$ instead of 1 allows us to conclude in the same way. □

For $\lambda \geq \sqrt{3} - 1$, and $m \geq 7$, it imposes a value of $\epsilon_\lambda \leq 0.323$, which gives a complexity multiplying factor only of 3.1 in the approximation of the knapsack problem.

4.5 Existence of a $2K$-schedule

In order to conclude that our algorithm based on the Knapsack is a $\sqrt{3}$-approximation for any instance of the malleable problem satisfying $S_m > \frac{\sqrt{3}}{2}m$, we have to establish the existence of a $2K$-schedule for such an instance. In the previous section we showed indeed that if a $2K$-schedule exists, solving the Knapsack problem $(\mathcal{P})$ finds an element of the set Θ_λ. We will prove in the following that for any value of $\lambda \geq \sqrt{3}-1$, the set $\Xi_\lambda \subseteq \Theta_\lambda$ has at least one element, except in some particular simple cases detailed below, which allows us to use the approximation scheme of section 4.4.

Trivial solutions. We start by particularizing some "trivial" solutions; basically it may happen that picking up one task of $\mathcal{S}_1$, all the other tasks (of $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$) can be scheduled in the first shelf, while this task enters the second shelf, i.e. requires at most m processors to be executed in time less than λ. We introduce $\bar{\Theta}_\lambda$ as the

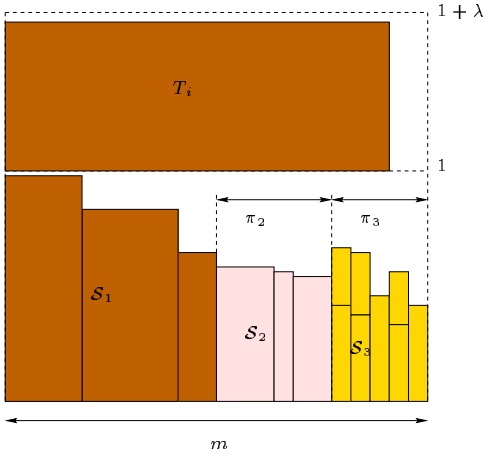

Figure 5: An example of a "trivial" solution of $\bar{\Theta}_\lambda$.

set of such tasks of $\mathcal{S}_1$. Hence $T_i \in \mathcal{S}_1$ belongs to $\bar{\Theta}_\lambda$ if it fulfills the two following conditions:

$$p_i \geq \pi_1 + \pi_2 + \pi_3 \quad \text{and} \quad q_i \leq m.$$

In addition we add to $\bar{\Theta}_\lambda$ all the one element sets belonging to Θ_λ. Clearly if $\bar{\Theta}_\lambda$ is not empty, one of its elements can be found in linear time in a preliminary phases of the algorithm, leading directly to a schedule of makespan at most $1+\lambda$.

2K-schedule solutions. Consider now, once again, the subset $\mathcal{A}$ of tasks of $\mathcal{S}_1$ executed in time less than $1/2$ in an optimal schedule σ. We denote by p the sum of the canonical number of processors of the tasks of $\mathcal{A}$, and q the analogous quantity when individually alloted to q_i processors. Our goal is to show that for $\lambda \geq \sqrt{3}-1$, either a subset of $\mathcal{A}$ belongs to Ξ_λ, either one of its elements is a trivial solution of $\bar{\Theta}_\lambda$. As noticed in section 4.2, we must have $p \geq \pi_1$ for σ to be feasible. By definition we also have $\sum_{\mathcal{A}} b_i \leq \mu_{\mathcal{A}} \sum_{\mathcal{A}} a_i$. Hence if q happens to be smaller than $m-\rho_2-\rho_3$, we can conclude that $\mathcal{A}$ belongs to Ξ_λ. This is certainly the case if the quantity $B = \sum_{\mathcal{A}} b_i$ is bounded by $\frac{\lambda}{2}(m-\rho_2-\rho_3)$; indeed for any task $T_i \in \mathcal{A}$ its number of processors q_i is greater than 2, which implies that $B > \sum_{\mathcal{A}} \frac{\lambda}{2} q_i = \frac{\lambda}{2} q$ due to property 1.

Thus assume in the following that we have $B > \frac{\lambda}{2}(m-\rho_2-\rho_3) = B_0$. Once again the master piece of the proof is the "control" of this optimal solution through its expansion factor $\mu_{\mathcal{A}}$. Intuitively a relatively important area of the tasks have been splashed in the optimal schedule to be executed in time less that $1/2$. Since the total canonical area of the instance is yet closed to m (recall that $S_m > (1+\lambda)/2m$), a huge splashed area B involves a small expansion factor $\mu_{\mathcal{A}}$ for the schedule σ to be feasible. We use the minoration $B > B_0$ to compute a more accurate upper bound for $\mu_{\mathcal{A}}$. From section 4.2 we have $\mu_{\mathcal{A}} \leq \frac{B}{A} \leq \frac{B}{S-m+B}$. This function is decreasing with B, and hence maximized for $B = B_0$. Using the different relations between the areas s_i and their corresponding number of processors, we get the majoration:

$$\mu_{\mathcal{A}} \leq \frac{\lambda}{(2\lambda-1)m + 2s_1 - \lambda}(m-\rho_2-\rho_3)$$

This more accurate majoration of the inefficiency factor $\mu_{\mathcal{A}}$ allows to establish the following lemma 3, which states that if the canonical area of a subset of $\mathcal{S}_1$ is bounded by $2\pi_1$, then the condition on the q_i's is fulfilled:

Lemma 3 *For all $\lambda \geq \sqrt{3}-1$ and all set $\zeta \subseteq \mathcal{S}_1$,*

$$\left\{ \begin{array}{l} \sum_\zeta a_i \leq 2\pi_1 - 1 \\ \sum_\zeta b_i \leq \mu_{\mathcal{A}} \sum_\zeta a_i \end{array} \right. \Rightarrow \sum_\zeta q_i \leq m - \rho_2 - \rho_3$$

Proof. Consider a subset ξ satisfying the conditions of the lemma. Since the relation $\sum_\xi q_i \leq \frac{2}{\lambda}\mu_{\mathcal{A}} \sum_\xi p_i$ holds, using the new majoration we have explicited for $\mu_{\mathcal{A}}$, we get:

$$\sum_\xi q_i \leq \frac{2}{\lambda}\frac{\lambda}{(2\lambda-1)m + 2s_1 - \lambda}(2\pi_1 - 1)(m-\rho_2-\rho_3)$$

Noticing that $\lambda\pi_1 \leq s_1$, we obtain the majoration $\sum_\xi q_i \leq g(s_1)\rho_1$, where $g(s_1)$ is defined by:

$$g(s_1) = \frac{2}{\lambda}\frac{2s_1 - \lambda}{(2\lambda-1)m + 2s_1 - \lambda}$$

This expression being an increasing function of s_1, and s_1 being bounded by $\frac{1-\lambda}{2}m$ by our assumption on the area S_m, we simply have to prove that $((1-\lambda)m - \lambda)/(\lambda m - \lambda)$ is lower than 1. Noticing that this expression is itself bounded by $(1-\lambda)/\lambda$ for any $\lambda \geq 1/2$, the majoration holds if λ verifies $\lambda^2 + 2\lambda - 2 \geq 0$. one can check easily that the polynomial function $X^2 + 2X - 2$ is positive for any real number greater than its root $\lambda_0 = \sqrt{3}-1$. □

To conclude the proof of the existence of a $2K$-schedule, we define the series $\mathcal{D}_0 = \mathcal{A} \subset \ldots \subset \mathcal{D}_i \subset \ldots \subset \emptyset$ where $\mathcal{D}_{i+1}$ is obtained from $\mathcal{D}_i$ removing the task of greatest inefficiency factor. Notice that by construction the inefficiency factors $\mu_{\mathcal{D}_i}$ form a non-increasing series, and hence are all bounded by $\mu_{\mathcal{D}_0} = \mu_{\mathcal{A}}$. It implies that any set $\mathcal{D}_i$ fulfills the second conditions of lemma 3. Notice also that the canonical area of $\mathcal{D}_i$ decreases with i, down to 0. Hence certainly one of the $\mathcal{D}_i$ fulfills both conditions of lemma 3, and reveals to be a good candidate to belong to Ξ_λ. This is stated by the following lemma:

Lemma 4 *For all $\lambda \geq \sqrt{3}-1$, if the set of the trivial solutions $\bar{\Theta}_\lambda$ is empty, then it exits an index i such that $\mathcal{D}_i \in \Xi_\lambda$.*

Proof. We prove the reverse implication of the lemma. Hence suppose that there does not exist any index in the series such that $\mathcal{D}_i \in \Xi_\lambda$. Due to our previous remark, as a corollary of lemma 3, it implies that the canonical area of any $\mathcal{D}_i$ is either greater than

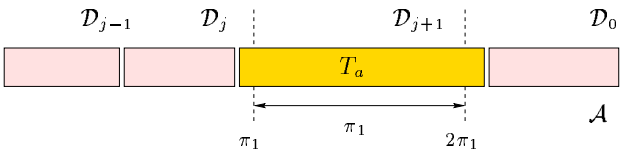

Figure 6: Representation of the series $(\mathcal{D}_i)_i$.

$2\pi_1$ or lower than $\pi_1 - 1$. Let j be the last index such that the area of $\mathcal{D}_j$ is greater than $2\pi_1$, and T_a be the task removed from $\mathcal{D}_j$ to $\mathcal{D}_{j+1}$. Necessarily T_a as a canonical area s_a greater than π_1, which implies that its canonical number of processors is also greater than π_1. Hence to prove that T_a belongs to the set of the trivial solutions $\bar{\Theta}_\lambda$, we just have to check that the minimal number q_a to execute it in time less than λ is lower than m. If $q_a > \rho_1$ then, by monotony, the corresponding execution area B_a is greater than $\lambda(q_a - 1) > \lambda\rho_1$. We use this majoration to determine once more a new bound on $\mu_{\mathcal{A}}$. We denote by B_b and s_b the area and canonical area of the set $\mathcal{D}_{j+1}$. Since the area of $\mathcal{A}$ is greater than $B_a + B_b$, we get:

$$\frac{B_a + B_b}{S - m + B_a + B_b} \;\geq\; \mu_{\mathcal{A}} \;\geq\; \mu_{\mathcal{D}_j} = \frac{B_a + B_b}{s_a + s_b}$$

As $B_a > \lambda\rho_1$ and $S = S_m + s_1 + s_2 + s_3$, we have:

$$s_a \;\geq\; \frac{3\lambda - 1}{2}m + s_1 + s_2 + s_3 - \lambda\rho_2 - \lambda\rho_3$$

To show that $T_a \in \bar{\Theta}_\lambda$, we just need to prove that $s_a \geq \pi_1 + \pi_2 + \pi_3$, or equivalently that the expression $\psi = s_a - (\pi_1 + \pi_2 + \pi_3)$ is positive. We use the inequalities $\lambda\pi_1 < s_1$, $\frac{1}{2}\pi_2 < s_2$ and $\frac{\lambda}{2}\rho_3 \leq s_3$.[3] We obtain:

$$\psi \;\geq\; \frac{3\lambda - 1}{2}m - (\frac{1}{\lambda} - 1)s_1 - (2\lambda + 1)s_2 - 3s_3$$

Due to our hypothesis on S_m, we have $s_1 + s_2 + s_3 \leq (1-\lambda)m/2$. The coefficient of the factors shows that the second term of the inequality is minimized taking $s_1 = s_2 = 0$ and $s_3 = (1-\lambda)m/2$. We get:

$$\psi \;\geq\; \frac{3\lambda - 1}{2}m - \frac{3}{2}(1 - \lambda)m \;=\; (3\lambda - 2)m$$

For $\lambda \geq 2/3$, this last expression is positive, which concludes the proof of lemma 4. □ Grouping the results of lemma 2 and lemma 4, and taking $\lambda = \sqrt{3} - 1$, we can state our final result:

Theorem 3 *For any instance admitting an optimal schedule lower than 1, such that the canonical m-area S_m is greater than $\sqrt{3}/2m$, a schedule of length at most $\sqrt{3}$ can be found in time complexity $\mathcal{O}(\min\{nm, n^3\} + n\log m)$, depending on the knapsack resolution algorithm (exact or approximate) used.*

5 Concluding Remarks

We have presented in this paper a new algorithm for scheduling a set of independent malleable tasks. It improves significantly the best bound known at this time, with a performance guarantee of $\sqrt{3}$. According to the value of the parameter S_m, depending on the canonical allotment of the instance, we solve the problem either using the canonical list scheduling algorithm in $\mathcal{O}(n\log m)$ or a knapsack based algorithm for the selection of the allotment in $\mathcal{O}(\min\{nm, n^3 + n\log m\})$. We would like to emphasis that the guarantee of $\sqrt{3}$ is a worst case performance bound not only on the instance, but also its parameters. Indeed the list scheduling algorithm has a guarantee $\frac{2S_m}{m}$ whatever the value of S_m. The restriction to get a better performance guarantee is that for $S_m < \sqrt{3}/2m$ the particular $2K$-schedule structure is not asserted to exist for the knapsack based algorithm. Nevertheless we have no doubt that in practise most instances admit such a schedule for better performance ratios than $\sqrt{3}$. Its existence can always be checked in time $\mathcal{O}(mn)$ solving the corresponding knapsack problem. Experiments are currently under progress to assert the good average behavior of our heuristics.

The natural continuation of this work is to study the scheduling of precedence graphs structures. Prasanna et al. [15] proved for the continuous version of the malleable tasks that the scheduling of general tasks graph has a very pleasant dominant structure: the number of processors allotted to the tasks can be seen as a flow through the precedence constraints of the graph. Such a structure permits to adapt an approximation for the independent malleable tasks to general precedence tasks graphs. We are also currently working on tree structures which occur on an actual parallel application for the simulation of the circulation in Atlantic ocean.

[3] The inequality $\frac{\lambda}{2}\rho_3 \leq s_3$ is not valid if $\rho_3 = \pi_3 = 1$. However for $m \geq 7$ it is easy to check in this case that ψ is positive.

Acknowledgment

The authors would like to thank an anonymous referee for his helpful remarks to improve the presentation of the paper.

References

[1] B.S. Baker, D.J. Brown, and H.P. Katseff. A 5/4 algorithm for two dimensional packing. *Journal of Algorithms*, 2:348–368, 1981.

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

[3] E. Blayo, L. Debreu, G. Mounie, and D. Trystram. Dynamic load balancing for ocean circulation model with adaptive meshing. *submitted to EuroPar'99*, 1999.

[4] R. Brent. *The Parallel Evaluation of Arithmetic Expressions in Logarithmic Time*, pages 83–102. Academic Press, New York, 1973.

[5] E.G. Coffman, M.R. Garey, D.S. Johnson, and R.E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, 1980.

[6] M.R. Garey and D.S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4, 1975.

[7] M.R. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W.H. Freeman, New York, 1979.

[8] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, March 1969.

[9] D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.

[10] K. Jansen and L. Porkolab. Linear time approximation schemes for scheduling problems. In *10th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 99*, pages 490–498, 1999.

[11] D.S. Jonhson, A. Demers, J.D. Ullman, M.R. Garey, and R.L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–329, December 1974.

[12] W. T. Ludwig. *Algorithms for scheduling malleable and non-malleable parallel tasks*. PhD thesis, University of Wisconsin - Madison, Department of Computer Sciences, 1995.

[13] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[14] G. N. S. Prasanna and B. R. Musicus. Generalised multiprocessor scheduling using optimal control. In 3*rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 216–228. ACM, 1991.

[15] G.N.S. Prasanna and B.R. Musicus. The optimal control approach to generalized multiprocessor scheduling. *Algorithmica*, 15(1):17–49, 1996.

[16] V.J. Rayward-Smith. UET scheduling with unit interprocessor communication delays. *Discrete Applied Mathematics*, 18:55–71, 1987.

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

[18] J. Turek, J. Wolf, and P. Yu. Approximate algorithms for scheduling parallelizable tasks. In 4*th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 323–332, 1992.

Appendix

A Proof of property 3

We present in appendix the proof of the property 3 of the canonical list algorithm, stating that any task of the second level completes before time 2λ. Recall that λ is an arbitrary real number of $]3/4, 1]$. For the property to be valid, we need the additional hypothesis that the area S_m is smaller than λm, refer to section 3.2 for all the definitions. We prove in the following the existence of a constant m_λ such that for any number of processors $m \geq m_\lambda$ the property holds.

Without loss of generality we can assume that the tasks are indexed by their decreasing execution times t_{i,p_i}. Due to the hypothesis on S_m, the first task, and the following ones, that can not be allocated to the first level of the algorithm has an execution time lower than λ. Consider the allocation of a task T_i on the second level, and let T_j be the task of the first level on which it is scheduled. For short we denote respectively by β and α their execution times (on their canonical numbers of processors). To prove the property 3 we have to establish that $\alpha + \beta \leq 2\lambda$. Recall that we have $\beta \leq \lambda$, hence to avoid trivial cases we can assume that $\alpha > \lambda$.

We denote by q the number of unoccupied processors on the first level at this step of the algorithm, cf figure 7. We also introduce p as the number of remaining processors on the right of T_j. The conservation of the work (property 2) and the condition on S_m provide us the following system equation on α and β:

$$(\mathcal{E}_q) \begin{cases} (m-p)\alpha + p\beta \leq \lambda m & \text{with } \beta \leq \alpha \leq 1 \\ (m-p)\alpha + (2p+1-q)\beta \leq m \end{cases}$$

To get a reasonable value for m_λ, we slightly modify our algorithm. Let q_λ be the greatest integer such that $q_\lambda/(q_\lambda+1) < \lambda$. On one hand, the monotony (property 1) of the tasks imposes that a task of duration lower than λ is allotted to at most $q_\lambda + 1$ processors. On the other hand allotting only $\bar{q}_\lambda = \lceil \frac{q_\lambda+1}{2} \rceil$ on such a task may at most double its execution time, hence still upper bounded by 2λ. Consider the situation when at the first allocation of a task T on the second level, at least $\bar{q}_\lambda$ processors remain unoccupied on the first level. We then decide to allocate in first place such a task T on $\bar{q}_\lambda$ processors on the first level. As we noticed task T completes before time 2λ, and necessarily after time 1. We call this case a *reallocation*. The rest of the tasks is then allocated using the list algorithm on $m_0 = m - \bar{q}_\lambda$ processors. For this new problem the area of the tasks is bounded by m_0, since T completes after time 1. In particular the lemma 1 still holds since only surface arguments are used. However the condition on S_{m_0} is a bit different from the one on S_m. Let $\beta_0 \leq \lambda$ be the execution time of T on its canonical number of processors, and $\beta \leq \beta_0$ the maximal execution time of the following tasks in the ordered list. Then we have:

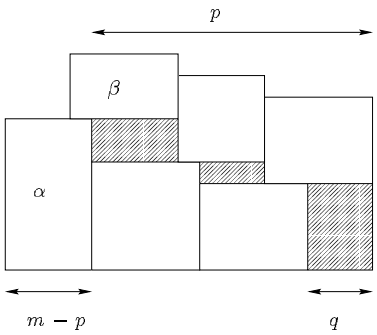

Figure 7: *Allocation of a task at the second level.*

$$S'_m \leq S_m - q\beta_0 + \bar{q}\beta \leq \lambda m_0 + (\lambda - \beta_0)\bar{q}_\lambda$$

Instead of $(\mathcal{E}_q)$ we hence consider the following system for the allocation of a task of length β on the second level:

$$(\mathcal{E}_{q,q_0}) \begin{cases} (m-p)\alpha + p\beta \leq \lambda m + (\lambda - \beta_0)q_0 \\ (m-p)\alpha + (2p+1-q)\beta \leq m \end{cases}$$

with $\beta \leq \beta_0,\ \beta \leq \alpha \leq 1$. Here q_0 plays the part of the (eventual) reallocated processors. Notice that if $q_0 = 0$, we have the initial inequality system $(\mathcal{E}_q)$ where no task is reallocated. From this system we will exhibit a constant $m(q, q_0)$ such that for $m > m(q, q_0)$ any pair of solution (α, β) verifies $\alpha + \beta \leq 2\lambda$. Writing m under the form $2p + k$, the first inequality gives:

$$p(\alpha + \beta) \leq 2\lambda p + q_0(\lambda - \beta_0) + k(\lambda - \alpha).$$

If we assume that $k \geq q_0$, since $\lambda - \alpha$ is a negative quantity, we have $p(\alpha + \beta) \leq 2\lambda p + q_0(2\lambda - \alpha - \beta)$, which implies

$(p+q_0)(\alpha+\beta) \leq 2(p+q_0)\lambda$. Hence $m \geq 2p+q_0$ (i.e. $k \geq q_0$) involves our result $\alpha+\beta \leq 2\lambda$. Assume on the contrary that $m \leq 2p+q_0$, and consider the second inequality. If the condition $m-p \leq 2p+1-q$ is satisfied, a simple exchange argument on the surface shows that $\alpha+\beta$ is maximized for $\alpha = 1$. We have then:

$$\beta \leq \frac{p}{2p+1-q}$$

To ensure that $\beta \leq 2\lambda - 1$ (i.e. $\alpha+\beta \leq 2\lambda$), it is sufficient to have:

$$p \geq \frac{2\lambda-1}{4\lambda-3}(q-1)$$

Since we are under the hypothesis $m \leq 2p+q_0$, the condition on p is satisfied for m greater than:

$$m_1(q,q_0) = \frac{4\lambda-2}{4\lambda-3}(q-1) + q_0 - 1.$$

Suppose now that we are in the other case where $m-p \geq 2p+1-q$. We write m under the form $m = 3p-q+1+k$ with $k \geq 0$. Recall that we have the additional hypothesis that $2p+q_0-1 \geq m$. It involves that $k \leq q_0+q-2-p$; in particular q_0 must be greater than 2 for this case to be considered. The second inequality of the system gives:

$$(2p-q+1)(\alpha+\beta) \leq (3p-q+1)+k(1-\lambda).$$

To ensure that $\alpha+\beta$ is lower than 2λ it is sufficient to have:

$$(1-\lambda)q_0 + \lambda q - 1 \leq (3\lambda-2)p.$$

This condition is fulfilled, since we have $m \leq 2p+q_0$, if m is greater than:

$$m_2(q,q_0) = \frac{\lambda}{3\lambda-2}(q_0+2q-3).$$

It results that we can assert $\alpha+\beta \leq 2\lambda$ for all $m \geq m(q,q_0) = \max\{m_1(q,q_0), m_2(q,q_0)\}$, the second term m_2 existing only for $q_0 \geq 2$.

In our algorithm, if no task is reallocated ($q_0 = 0$), we must verify the system $(\mathcal{E}_{q,0})$ with $q < \bar{q}_\lambda$, i.e. with possibly $\bar{q}_\lambda - 1$ idle processors at the first level. On the contrary if a task is reallocated, we have $q_0 = \bar{q}_\lambda$, and at most $q_\lambda - \bar{q}_\lambda$ processors can remain unoccupied on the first level. We must then satisfy the system $(\mathcal{E}_{q,\bar{q}_\lambda})$, with $q \leq q_\lambda - \bar{q}_\lambda$. Hence we set

$$m_\lambda = \max\{\, m_1(\bar{q}_\lambda - 1, 0),\ m(q_\lambda - \bar{q}_\lambda, \bar{q}_\lambda)\,\}$$

The figure 8 shows the value of m_λ according to λ. For $\lambda = \sqrt{3}/2$, the value we are interested in, the previous expression gives $\lceil m_\lambda \rceil = 8$. However we can refine our analysis to prove that the property holds for $m = 7$. In this case, since the canonical number of processors of task is at most 7, we have to consider $q_\lambda = 6$, involving $\bar{q}_\lambda = 4$. In the expression of m_λ, the values $m_1(3,0)$ and $m_1(2,4)$ are lower than 7; the last value $m_2(q,4)$ being strictly greater than 7 only for $q \geq 2$. Hence the only case when $\lceil m_\lambda \rceil$ is greater than 7 (in fact equals to 8) happens for a task T reallocated to 4 processors remaining 2 processors idle on the first level. However at least one task have been scheduled before the reallocation of T. If 2 processors remain idle this task occupy only one processor, while T was initially alloted to 7 processors (otherwise T can be scheduled without reallocation). Due to property 1 it involves a canonical execution time of T greater than $6/7$, and so a work area for the two tasks altogether greater than $8 \times 6/7 \simeq 6.86$.

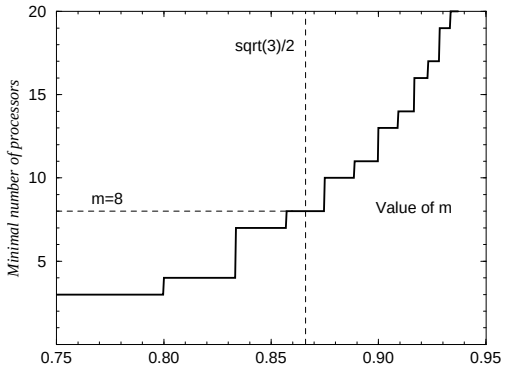

Figure 8: *Value of $\lceil m_\lambda \rceil$ in function of λ.*

The total area of the remaining tasks being less than 0.15, the property 1 shows that any of these tasks is necessarily allotted to one processor, with an execution time off course lower than 0.15. Clearly they are all scheduled by the *LPT* rules on the two remaining processors and complete before time 1. It permits us to conclude that the schedule length can not exceed $\sqrt{3}$.

B Proof of lemma 1

We present here the proof of the lemma 1 of the section 3.2. Consider a task T_i scheduled outside the first two levels by the canonical list algorithm. We establish here that T_i is then a sequential task (alloted to one processor) of execution time lower than $1/2$. For short we denote by p and t respectively its canonical number of processor p_i and its execution time t_{i,p_i}. We start by proving that necessarily all processors compute a task of the two first levels, i.e. that these levels are totally occupied. We denote by q and q' the number of processors unoccupied respectively on the first and the second level of the schedule when task T_i is allocated. The only reason for T_i not to be allocated in this two levels is that $k = \max\{q, q'\}$ is strictly lower than p. Using the property 1, the execution time t of T_i is then greater than $k/(k+1)$ and its area greater than k. However the total area of the tasks yet allocated to the first two levels is at most $(2m-k)t$. Writing down the property 2, we get:

$$m \geq (2m-2k)t + k > \frac{2k}{k+1}(m-k) + k$$

It implies that $1 > 2k(k+1)$, and so $k = 0$. It shows that the first two levels are totally occupied. Rewriting the property on the surface, we have $m \geq (2m+1)t$. It involves that the execution time of the task is strictly lower than $1/2$, and thus, due to monotony, it is executed on only one processor.