A best-compromise bicriteria scheduling algorithm for malleable tasks

Pierre-François Dutot, Denis Trystram

A BEST-COMPROMISE BICRITERIA SCHEDULING ALGORITHM FOR PARALLEL TASKS

Pierre-François Dutot and Denis Trystram
ID-IMAG - 51, avenue J. Kuntzmann
38330 Montbonnot St-Martin, France
pfdutot@imag.fr

Abstract

We consider in this paper the problem of scheduling a set of independent parallel tasks (jobs) with respect to two criteria, namely, the makespan (time of the last finishing job) and the minsum (average completion time). There exist several algorithms with a good performance guaranty for one of these criteria. We are interested here in studying the optimization of both criteria simultaneously. The numerical values are given for the moldable task model, where the execution time of a task depends on the number of processors alloted to it. The main result of this paper is to derive explicitly a family of algorithms guaranteed for both the minsum and the makespan. The performance guaranty of these algorithms is better than the best algorithms known so far. The *Guaranty curve* of the family is the set of all points $(x; y)$ such that there is an algorithm with guarantees x on makespan and y on the minsum. When the ratio on the minsum increases, the curve tends to the best ratio known for the makespan for moldable tasks (3/2). One extremal point of the curves is a (3;6)-approximation algorithm. Finally a randomized version is given, which improves this results to (3;4.08).

Keywords: Multi-criteria Analysis, Packing, Scheduling, Parallel Processing.

1. Introduction

The most important results about scheduling have been established for the makespan criterion (which gives the time of the last finishing job). In the context of the new execution supports like clusters or grid-computing [3], tasks correspond to independent programs that are submitted by different users at any time. Other objectives have to be considered for optimizing the management of the resources.

The Parallel Tasks model (tasks that require one or more processors for their execution) has been introduced about 15 years ago [4]as a promising alternative for scheduling parallel applications, especially in the case of slow communication media. The basic idea is to consider the application at a rough level of granularity (larger tasks in order to decrease the relative weight of communications).

Informally, a Parallel Task (PT) is a *task* (sometimes also called a *job*) that gathers elementary operations, typically a numerical routine or a nested loop, which contains itself enough parallelism to be executed by more than one processor. More recently, the Moldable Task (MT) model has been introduced [15]to extend the PT model. A MT is a PT which can be alloted to any number of processors, its execution time depending on the number of processors.

The mathematical definition of an "efficient" schedule is a schedule that minimizes a given objective function. Several objective functions have been proposed and studied in the past, we will focus here on the makespan and minsum criteria. The makespan is the most popular in a single user framework, where the final result is only available when all tasks have been completed, whereas the minsum criterion is generally related to a multi-user execution scheme, where the tasks are competing for the computing resources. In the minsum case, giving different weights to the tasks determines which tasks are more important and have to be executed quickly and which tasks may be delayed for a later execution.

In a multi-user framework, the tasks also often arrive at any time during the schedule, and are only known when they arrive. These jobs are "on-line" as opposed to the "off-line" jobs which are available at any times.

An efficient way of dealing with on-line jobs is to consider batch executions: the jobs are gathered into sets (called batches) that are scheduled together. All further arriving jobs are delayed to be considered in the next batch. This is a nice way for dealing with on-line algorithms by a succession of off-line problems. Shmoys et al. [13]proposed a construction to adapt an algorithm for scheduling independent tasks without release dates (all tasks are available at date 0) with a performance ratio of ρ on the makespan into a batch scheduling algorithm with unknown release dates with a performance ratio of 2ρ on the makespan.

The main result of this paper is the construction of a family of algorithms which optimize together two criteria. Each algorithm of this family with guaranty ρ_1 on the makespan and ρ_2 on the average completion time (denoted $(\rho_1;\rho_2)$-approximation algorithm) corresponds to a point on the *Guaranty* curve of the family (the curve of the performances

of the algorithms for which there are no known algorithm better on both criteria).

The paper is organized as follows: In the next section we give some basic notations and definitions used throughout the paper. We recall in Section 3 the most significant existing results for each of the criteria. Then, in Section 4 an existing bicriteria algorithm is presented briefly. In Section 5, we describe our family of algorithms and the associated performance ratios. An interesting randomization scheme applied in Section 6 yields even better results. We conclude with a short discussion on on-going works.

2. Notations and basic definitions

We consider n independent jobs executed on m identical processors.

The execution time of task i ($1 \leq i \leq n$) is denoted $p_i(q)$ when it is allocated to q processors ($q \leq m$). The starting time of task i is $\sigma(i)$, and its completion time is $C_i = \sigma(i) + p_i(q)$. When needed, the number q of processors used by task i will be given by $q = nbproc(i)$.

We will restrict the analysis on Moldable Tasks that start their execution on all processors simultaneously. In other words, the execution of moldable tasks corresponds to rectangles when scheduled on contiguous processors. We also use a common hypothesis in Parallel Processing, the monotony, which states that when allotting more processors to a task its execution time decreases while its work increases [9].

The MTS (Moldable Tasks Schedule) problem with one criterion (i.e. one objective function f to minimize) is defined as follows:
Instance: A set of n weighted moldable tasks defined by their respective weights ω_i and their respective execution times $p_i(q)$ for $1 \leq q \leq m$ and $1 \leq i \leq n$ (depending on the number of processors alloted to the task).
Question: Determine a feasible schedule which minimizes the objective function f. In the next section, we will discuss the two functions used here.

A feasible schedule is a pair of functions $(\sigma, nbproc)$ of $V \rightarrow N \times [1..m]$ (σ is the starting time and $nbproc$ the number of processors), such that no more than m processors are used at any time slot.

With two criteria, there is two objective functions to minimize. In this case there is sometimes no solution which is optimal for both criterion.

3. Single criterion

Among all the possible criteria found in the literature, two are really relevant for the moldable tasks scheduling problem:

- Minimisation of the *makespan* (completion time $C_{max} = max(C_i)$ where $C_i = \sigma(i) + p_i(nbproc(i)))$
- Minimisation of the average completion time ΣC_i [12, 1]and its variant average weighted completion time ($\Sigma \omega_i C_i$). The weight is used for priority: tasks with a bigger weight will generally be executed earlier than the others.

3.1 Optimizing the makespan

Scheduling is a question of "where" and "when". However with moldable tasks, the question "where" is a double question: "how many processors" and "which processors". The general approach for most existing algorithm is to separate these two questions into two different phases, as when the allocation is fixed the MT problem becomes a regular PT problem (also called "rigid" because the number of processors is fixed) usually approximated with a strip packing algorithm.

The idea that has been introduced in [15]is to optimize in the first phase (choosing the number of processors needed by each task) the criterion used to evaluate the performance ratio of the second phase (the scheduling of the now "rigid" tasks). The authors proposed to realize a trade-off between the maximum execution time (critical path) and the sum of the works.

The performance ratio of this algorithm is fixed by the corresponding strip packing algorithm (or whatever rigid scheduling algorithm used in the second phase). As such problems are NP-hard, the only way to obtain better results is to solve different allocation problems which lead to "easier" scheduling problems.

The idea is to determine the task allocation with great care in order to fit them into a particular packing scheme. This approach where the allocation phase prepares the scheduling phase was given in [10]. The resulting algorithm has a performance ratio of $3/2 + \epsilon$. It is obtained by stacking two shelves of respective sizes λ and $\frac{\lambda}{2}$ where λ is a guess of the optimal value C^*_{max}. This guess is computed by a dual approximation scheme [8]. Informally, the idea behind dual approximation is to fix a hypothetical value for the guess λ and to check if it is lower than the optimal value C^*_{max} by running a heuristic with a performance ratio equal to ρ and a value C_{max}. If $\lambda < \frac{1}{\rho} C_{max}$, by definition of the performance

ratio, λ is underestimated. A binary search allows us to refine the guess with an arbitrary accuracy ϵ.

3.2 Optimizing the average completion time

For a first view of the problem, we present the principle of the algorithm from Schwiegelshohn et al. [11]which is dedicated to this criterion.

The "smart SMART" algorithm of [11]is a shelf algorithm. It has a performance ratio of 8 in the unweighted case and 8.53 in the weighted case ($\sum \omega_i C_i$). The "height" (time length) of a shelf is a power of two in the unweighted case, and a power of 1.65 in the weighted case. All tasks are packed (first fit, largest area first) into one of the shelves just sufficient to include it. Then all shelves are sorted in order to minimize $\sum \omega_i C_i$, using a priority of $\frac{H_l}{\sum_l \omega_i}$, where H_l is the height of shelf l.

The basic point of the proof is that the shelves may be partitioned in two sets: a set including exactly one shelf of each size, and another one including the remaining shelves. Their completion times are bounded by two different lower bounds of the optimal minsum. The combination can be adjusted to get the best performance ratio.

4. Existing Multicriteria result

Schedules created for the C_{max} criterion and schedules created for the $\sum \omega_i C_i$ are generally very different. Actually, when evaluated according to the other criterion they usually perform very badly. In a setting where the two criteria are important, an algorithm which does not specifically minimize one of these criteria but gives an acceptable solution for both is often preferred. Optimizing simultaneously several criteria received a lot of interest recently. For a complete survey in scheduling see the recent book of T'kindt and Billaut [14].

A first approach is to merge two existing algorithms, each dedicated to a criterion. An alternative approach has been considered by Hall et al. [6](see [7]for a longer version). Their scheme is based on an off-line scheduling algorithm for the makespan from which they derive an on-line algorithm for the minsum case.

The main idea here is to create a batch schedule which has a good performance ratio on the sum of completion times without losing too much on the makespan. We will briefly present this algorithm with the same notation as in [6], as it is closely related to our best compromise algorithm.

The algorithm is based on a dual ρ-approximation algorithm for the Maximum Scheduled Weight Problem (MSWP), which is defined as follows: Given a set of jobs available at time 0, a weight for each job and

a deadline D, build a schedule that maximizes the total weight of jobs completed by time D.

A ρ-approximation algorithm for this problem builds a schedule of length ρD with at least as much weight as the optimal in D.

The time horizon is then split at geometrically increasing points. Assuming without loss of generality that no tasks has an execution time smaller than 1, let $\tau_1 = 1$ and $\tau_l = 2^{l-1}$. The schedule is constructed iteratively. At step l, we wait until τ_l and denote J_l the set of jobs available at that time. The off-line MSWP algorithm is then run on the set J_l with deadline $D = \tau_l$, and the result scheduled between time $\rho\tau_l$ and $\rho\tau_{l+1}$.

At this point a very useful dominance property can be proven. Let $\bar{S}_l$ and $\bar{W}_l$ denote respectively the set of jobs scheduled by the MSWP algorithm during iteration l and their total weight. And let S_l denote the set of jobs that complete in the interval $(\tau_{l-1}, \tau_l]$ and W_l the total weight of S_l. The set $S = (\cup_{k=1}^{l} S_k) - (\cup_{k=1}^{l-1} \bar{S}_k)$ is the set of tasks completed in the considered optimal schedule before time τ_l but not completed by the algorithm before time $\rho\tau_l$. Clearly $S \subseteq J_l$, and can be scheduled in τ_l units of time (as it is in the optimal). Therefore the weight scheduled by the MSWP algorithm in step l is at least equal to the weight of S. Which implies the dominance property that for each l we have $\sum_{k=1}^{l} \bar{W}_k \geq \sum_{k=1}^{l} W_k$.

Let L be defined so that each job completes by time τ_l, the sum of weighted completion times of the constructed schedule is lower than $\sum_{l=1}^{L} \rho\tau_{l+1}\bar{W}_l$. The following inequalities allows us to conclude on the performance ratio for the minsum criterion:

$$\sum_{l=1}^{L} \rho\tau_{l+1}\bar{W}_l \leq 4\rho \sum_{l=1}^{L} \tau_{l-1}\bar{W}_l \leq 4\rho \sum_{l=1}^{L} \tau_{l-1}W_l \leq 4\rho \sum_{j=1}^{n} \omega_j C_j^*$$

For the makespan criterion, the performance ratio is also 4ρ as the last task to complete in the optimal completes in the interval $(\tau_{L-1}, \tau_L]$ and in the constructed schedule, the last task completes in the interval $(\rho\tau_L, \rho\tau_{L+1}]$.

These ratios are worst case ratios. For mean performance ratios, a randomization technique improves the results to $\frac{2}{\ln(2)}\rho$ for both criteria.

5. Best compromise

5.1 A better MSWP algorithm

In [2], the authors use a 3-approximation algorithm for the Maximum Scheduled Weight Problem. However this result can be greatly

improved using the best off-line algorithm for scheduling moldable tasks with respect to the makespan criterion.

The algorithm given in [10]makes a selection in the allocation phase to sort the tasks in two sets S_1 and S_2 scheduled differently. We propose to modify this selection to allow the rejection of tasks. Instead of splitting in two sets, we split the set of tasks in three sets: S_1, S_2 and the rejected ones. With integer weights, the complexity of the algorithm is multiplied by $n\omega_{max}$ where n is the number of tasks and ω_{max} the maximum weight.

With this new algorithm, the performance ratio is halved: from $12+\epsilon$ to $6+\epsilon$.

5.2 An off-line family of algorithms

The previous algorithm was clearly designed for an on-line setting with release dates. However, it can be transformed in a family of off-line algorithms with better performance ratios.

We detail below how the off-line schedules are constructed. Let us now consider that we know[1] an optimal schedule for the $\sum C_i$ criterion. For a given α we can transform this schedule into a simpler but less efficient schedule as follows:

1 Let C^*_{max} be the optimal makespan for the considered instance. Let k be the smallest integer such that in the considered $\sum C_i$ schedule, there is no task finishing before $\frac{C^*_{max}}{\alpha^k}$.

2 All the tasks i with $C_i < \tau_1 = \frac{C^*_{max}}{\alpha^{k-1}}$ can be scheduled in $\rho\frac{C^*_{max}}{\alpha^{k-1}}$ units of time, as $\frac{C^*_{max}}{\alpha^{k-1}}$ is the makespan of a feasible schedule for the instance reduced to these tasks, therefore larger than the optimal makespan for the reduced instance. These tasks are scheduled within time $\bar{\tau}_1 = \rho\frac{C^*_{max}}{\alpha^{k-1}(\alpha-1)}$ and time $\bar{\tau}_2 = \bar{\tau}_1 + \rho\frac{C^*_{max}}{\alpha^{k-1}} = \rho\frac{C^*_{max}}{\alpha^{k-2}(\alpha-1)}$.

3 Similarly for the remaining intervals ($j = k-2$ down to 1), all the remaining tasks i with $C_i < \tau_{k-j} = \frac{C^*_{max}}{\alpha^j}$ can be scheduled in $\rho\frac{C^*_{max}}{\alpha^j}$ units of time, right after the tasks already scheduled. Hence between $\bar{\tau}_{k-j} = \rho\frac{C^*_{max}}{\alpha^j(\alpha-1)}$ and $\bar{\tau}_{k+1-j} = \rho\frac{C^*_{max}}{\alpha^{j-1}(\alpha-1)}$

4 All the remaining tasks can be scheduled in ρC^*_{max} units of time, as the optimal value of the makespan is C^*_{max}. Again they are placed right after the previous ones.

[1]This assumption is not needed by the algorithm, it is only made to provide an insight of the construction.

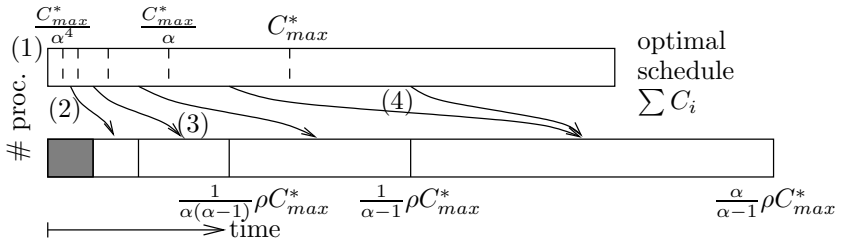

Figure 1. Transformation of an optimal schedule for $\sum C_i$ in a bi-criteria schedule (with $k = 4$ and $\alpha = 2$).

The transformation and resulting schedule is shown in Figure 1. The number within parentheses refer to the four steps of the transformation written above. If $C_i^{\mathbf{s}}$ are the completion times in the schedule before the transformation and $C_i^{\mathbf{t}}$ are the completion times after it, we can say that for all tasks i such that $\frac{C^*_{max}}{\alpha^j} < C_i^{\mathbf{s}} \leq \frac{C^*_{max}}{\alpha^{j-1}}$ we have $\rho\frac{C^*_{max}}{\alpha^{j-1}(\alpha-1)} < C_i^{\mathbf{t}} \leq \rho\frac{C^*_{max}}{\alpha^{j-2}(\alpha-1)}$ in the transformed instance, which means that $C_i^{\mathbf{t}} < \frac{\alpha^2}{\alpha-1}\rho C_i^{\mathbf{s}}$. With this transformation the performance ratio with respect to the $\sum C_i$ criterion is $\frac{\alpha^2}{\alpha-1}\rho$ and the performance ratio to the C_{max} criterion is $\frac{\alpha}{\alpha-1}\rho$. A simple function study shows that interesting values are for $\alpha \geq 2$.

Description of the algorithm As we are working on an off-line problem, a binary search with the ρ MSWP algorithm gives us a lower bound $\tilde{C}_{max}$ on the optimal makespan. With the tasks execution times and this lower bound, for a given α we compute the smallest k such that no task can complete in less than $\frac{\tilde{C}_{max}}{\alpha^k}$. We construct the schedule iteratively, running at step j the MSWP algorithm on all the remaining tasks with deadline $D_j = \frac{\tilde{C}_{max}}{\alpha^{k-j}}$ and placing them as in the previous transformation. As we know from the binary search that all the tasks can be scheduled by the ρ MSWP algorithm within $\rho\tilde{C}_{max}$ units of time, all the remaining tasks are scheduled in step k and the makespan is at most $\frac{\alpha}{\alpha-1}\rho C^*_{max}$. For the minsum, the dominance property proven in Section 4 still applies here. Therefore the performance ratios of these algorithms are $\frac{\alpha}{\alpha-1}\rho$ on the makespan and $\frac{\alpha^2}{\alpha-1}\rho$ on the minsum criterion.

6. Randomization

To further improve these results, we can also adapt the randomization technique presented in [2]. One of the drawback of the worst case performance ratio is that it does not necessarily reflect the behavior of the algorithm on the practical set of instances on which the algorithm

will be used. In the worst case the tasks are ending just after one of the τ_j in the optimal schedule, and are ending just before $\bar{\tau}_{j+2}$ in our schedule. If the $\bar{\tau}_j$ (and τ_j) is multiplied by a constant factor β chosen between $(1/\alpha, 1]$, the instances with the worst ratio depend on β. This means that running the algorithm with some values of β can provide different schedules with different values of the objective functions. We are interested here in the average performance ratio when the parameter β is taken between $(1/\alpha, 1]$. As β can be lower than 1, we will have to add an extra batch of length ρC^*_{max} at the end of the schedule. This extra batch will have no effect on the performance ratio of the minsum criterion, however it will increase the ratio of the makespan by ρ.

Considering an optimal schedule for the minsum criterion, let $\tau_{end(i)}$ be the beginning of the interval where task i finishes ($\tau_{end(i)} \leq C^*_i < \tau_{end(i)+1}$). With the notations of Section 4, the sum of completion times of the schedule is still bounded by $\sum_{l=1}^{L} \bar{\tau}_{l+1}\bar{W}_l$. As previously we can write the following inequalities:

$$\sum_{l=1}^{L} \bar{\tau}_{l+1}\bar{W}_l \leq \alpha^2 \sum_{l=1}^{L} \bar{\tau}_{l-1}\bar{W}_l \leq \rho\frac{\alpha^2}{\alpha-1} \sum_{l=1}^{L} \tau_{l-1}\bar{W}_l \leq \rho\frac{\alpha^2}{\alpha-1} \sum_{l=1}^{L} \tau_{l-1}W_l$$

But this time we will not overestimate $\sum_{l=1}^{L} \tau_{l-1}W_l$ by $\sum_{i=1}^{n} \omega_i C^*_i$, but more precisely by $\sum_{i=1}^{n} \omega_i \tau_{end(i)}$.

$$\sum_{l=1}^{L} \bar{\tau}_{l+1}\bar{W}_l \leq \rho\frac{\alpha^2}{\alpha-1} \sum_{i=1}^{n} \omega_i \tau_{end(i)}$$

This time the last inequality does not refer to the completion times of the tasks in an optimal schedule, but to the beginning of the intervals where the tasks complete. With $\beta = \alpha^{-X}$ where X is a random variable uniformly distributed in the interval $[0, 1)$, the expectation of $\tau_{end(i)}$ is:

$$E[\tau_{end(i)}] = C^*_i \int_0^1 \alpha^{-x} dx = \frac{\alpha-1}{\alpha \ln(\alpha)} C^*_i$$

The mean ratio on the minsum is therefore $\frac{\alpha}{\ln(\alpha)}\rho$. With the extra batch, the ratio on the makespan is now at most $\frac{2\alpha-1}{\alpha-1}\rho$. The average ratio for the makespan is $(1 + \frac{1}{\ln(\alpha)})\rho$, but there may be no value of β for which both average are reached.

Figure 2 depicts the curves of average minsum with worst-case makespan (as a solid line) and average minsum with average makespan (as a dashed line). For comparison, the curve of worst-case ratios from the previous section is also drawn (as a dotted line). For the deterministic version of

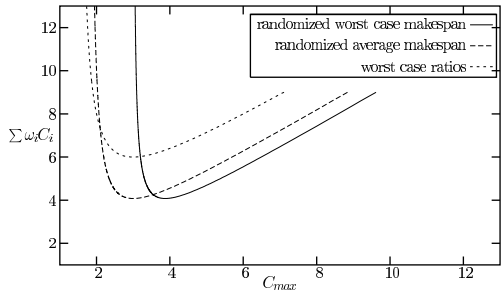

Figure 2. Comparison of the different algorithms

the algorithm, the minimum guaranty for minsum is achieved for $\alpha = 2$. For smaller values of α the algorithms guaranties are worse than this minimum value, whereas for larger values of α, the minsum guaranty is worse. However, the makespan guaranty is improved.

Remark that the minimum value of the average ratio is reached when $\alpha = e \simeq 2.72$ instead of $\alpha = 2$ in the previous case. This value is $e\rho$, which gives a performance ratio of approximately 4.08 (taken $\rho = 3/2$) on the minsum criterion for our problem of scheduling moldable tasks, with a worst case ratio of 3.88 on the makespan and an average ratio of 3.

7. Concluding remarks

We presented in this paper a family of algorithms for scheduling moldable tasks off-line with a good performance ratio on both the makespan and the minsum criteria. The ratios obtained by these algorithms are such that there is no known algorithm which is better than one of the family on both criteria at the same time.

This improvement is built on previous work from [2], improving their results in three ways. The first improvement is due to the off-line setting of the work presented here which reduces the guarantee on the makespan by a factor 2. The second improvement is to provide a better ρMSWP algorithm which improves both guarantees by another factor 2. The last improvement is to consider a parameter α instead of doubling the size of the batches. This last improvement provides a full family of algorithms, and shows that in the randomized case the previous results from [2]are completely dominated by the algorithm with $\alpha = 2.72$.

From this work, we derived some simplified algorithms that we tested on realistic workloads. Results from these experiments have already been published [5].

References

[1] F. Afrati, E. Bampis, A. V. Fishkin, K. Jansen, and C. Kenyon. Scheduling to minimize the average completion time of dedicated tasks. *Lecture Notes in Computer Science*, vol. 1974, 2000.

[2] S. Chakrabarti, C.A. Phillips, A.S. Schulz, D.B. Shmoys, C. Stein, and J. Wein. Improved scheduling algorithms for minsum criteria. *Lecture Notes in Computer Science*, (1099):646–657, 1996.

[3] D. Culler, J. Singh, and A. Gupta. *Parallel Computer Architecture - A Hardware,Sofware approach*. Morgan Kaufmann Pub., 1999.

[4] J. Du and J.Y-T. Leung. Complexity of scheduling parallel tasks systems. *SIAM Journal on Discrete Mathematics*, 2(4):473–487, November 1989.

[5] Pierre-François Dutot, Lionel Eyraud, Grégory Mounié, and Denis Trystram. Bi-criteria algorithm for scheduling jobs on cluster platforms. In *Sixteenth ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'04)*, Barcelona, Spain, June 2004.

[6] L. Hall, D. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. In *Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms*, pages 142–151, 1996.

[7] L. A. Hall, A. S. Schulz, D.B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22(3):513–544, 1997.

[8] 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.

[9] R. Lepère, G. Mounié, D. Trystram, and B. Robič. *Parallel Computing, Fundamentals and Applications*, chapter Malleable Tasks: An efficient model for solving actual parallel applications, pages 598–605. Imperial College Press, August 1999.

[10] G. Mounié. *Ordonnancement efficace d'application parallèles : les tâches malléables monotones*. PhD thesis, INP Grenoble, juin 2000.

[11] U. Schwiegelshohn, W. Ludwig, J. Wolf, J. Turek, and P . Yu. Smart SMART bounds for weighted response time scheduling. *SIAM Journal on Computing*, 28, 1998.

[12] H. Shachnai and J. Turek. Multiresource malleable task scheduling to minimize response time. *Information Processing Letters*, 70:47–52, 1999.

[13] D.B. Shmoys, J. Wein, and D.P. Williamson. Scheduling parallel machine online. *SIAM Journal on Computing*, 24(6):1313–1331, 1995.

[14] V. T'kindt and J.-C. Billaut. *Multicriteria Scheduling – Theory, Models and Algorithms*. Springer Verlag, 2002.

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