Approximation Algorithms for Scheduling Independent Malleable Tasks[⋆]

J. Błażewicz, M. Machowiak[1], G. Mounié, and D. Trystram[2]

[1] Instytut Informatyki Politechnika Poznanska
ul. Piotrowo 3a, 60 - 965 Poznan, Poland
[2] ID-IMAG, 51 rue Jean Kuntzman
38330 Montbonnot Saint Martin, France

Abstract. Malleable tasks consist in considering the tasks of a parallel program as large computational units that may be themselves parallelized. In this paper we investigate the problem of scheduling a set of n independent malleable tasks on a m processors system, starting from the continuous version of the problem.

1 Introduction

The malleable task model is a recent model in parallel processing introduced in order to solve efficiently some practical problems [5,6,7]. These problems have complex behavior at the finest level of execution which brings classical methods of scheduling to their limits, mainly due to the explicit management of the communications. The idea of a malleable task (MT) results in solving the problem at a different level of granularity in order to globally take into account communication costs and parallelization overheads with a simple penalty factor.

Malleable tasks can be distinguished from multiprocessor tasks, considered for example in [1], where the number of processors allotted to each task is known. The latter model has received a considerable attention in the literature. The problem of scheduling independent MT without preemption (it means that each task is computed on a constant number of processors from its start to completion) is NP-hard [2], thus, an approximation algorithm with performance guarantee has been looked for. While the problem has an approximation scheme for any fixed value m, the number of processors, [4], no general practical polynomial approximation better than 2 is known [5].

In this paper starting from the continuous version of the problem (i.e. where the tasks may require a fractional part of the resources), we propose a different approximation algorithm with a performance guarantee equal to 2. Then, some improvements are derived.

[⋆] This work was realized when J. Błażewicz was visiting ENSGI, Grenoble, in Spring 2000 and was partially supported by KBN Grant 8T11A01618

2 Problem Formulation

We consider a set of m identical processors $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ used for executing the set $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ of n independent, non-preemptable malleable tasks (MT). Each MT needs for its execution at least 1 processor. The number of processors allotted to a task is unknown in advance. The *processing speed* of a task depends on the number of processors allotted to it: namely, function f_i relates processing speed of task T_i to a number of processors allotted. The criterion assumed is schedule length.

Now, the problem may be stated as the one of finding a schedule (a processor allocation to tasks) of minimum length ω, provided that the processing speed functions of the tasks are all *concave*. Let us note, that this is a realistic case, more often appearing in practice. We will denote this minimum value as $\omega_m^\star$.

As far as processors are concerned, functions f_i are discrete. However, in general, it would be also possible that these functions are continuous. In what follows we will use the results of optimal continuous resource allocation to construct good processor schedules. The reader will find the basic results from the optimal continuous resource allocation theory in [10]. To distinguish it from a discrete case, an optimal schedule length of a continuous case will be denoted by C^*_{cont}. Basically, in the continuous case, all tasks are executed from time 0 to C^*_{cont} on a fraction of the available processors.

3 An Approximation Algorithm and Its Worst Case Behavior

In this section we propose an algorithm to transform a schedule obtained from the continuous version into a feasible schedule for the discrete MT model and prove that it has a performance guarantee of 2.

Every task T_i with an allocation $r_i \geq 1$ in the continuous solution is scheduled on $\tilde{r}_i = \min(r, t_i(r) \leq 2 \times C^*_{cont})$ processors in the malleable scheduling. All parallel tasks (allotment strictly larger than 1) start at time 0. Other tasks, for which a continuous allotment $r_i < 1$, receive one processor and are scheduled in free time slots (cf Algorithm 1).

Algorithm 1 Generic transformation algorithm.

```
Compute $C^*_{cont}$ and $\forall i, r_i$
for $i = 1$ to $n$, $\tilde{r}_i = \min(r, t_i(r) \leq 2 \times C^*_{cont})$
for $i = 1$ to $n$
    if $r_i < 1$ then $\tilde{r}_i = 1$
    else if $\tilde{r}_i > 1$ then $Start(i) = 0$
for $i = 1$ to $n$
    if $\tilde{r}_i = 1$ then
        $Start(i) = MinimumDateAvailableProcessor()$
```

Note, that the complexity of the above transformation algorithm is $O(n\log(m))$, as it required to maintain a heap of processor load.

Theorem 1 *Algorithm 1 has a performance guarantee of* 2*.*

Proof The continuous solution consists in executing simultaneously all the tasks on a fractional number of processors. This solution realizes the trade-off between the total work (task duration by allotted processors) and the length of the tasks. Thus C^*_{cont}, is a lower bound on ω^*_m.

For all the tasks which continuous allotment $r_i \geq 1$, by construction of the Algorithm 1, their duration are less than 2 C^*_{cont}. Their work decreases. Moreover, the sum of the processors allotted to these tasks after the transformation stays lower than m. Thus, these tasks, whose duration is between C^*_{cont} and 2 C^*_{cont}, can be executed on less than m processors starting at time 0, and their work is smaller than mC^*_{cont}.

The tasks which continuous allotment $r_i < 1$ are assigned on one processor. Their execution times decrease but their work increases. This surface is still the minimal one that these tasks can have in any discrete malleable schedule. Thus these tasks have a duration lower than C^*_{cont} and their total surface is less than $m\ \omega^*_m$.

The sum of the surfaces of the tasks is lower than $m\ C^*_{cont}+m\ \omega^*_m$. An analysis similar to Graham's one can be now applied [3]. The last task that is allotted starts at a time when all the processors are busy (otherwise, it could have been started earlier). Thus, the schedule length of the malleable (discrete) schedule, ω_m is lower than $max\{2\ C^*_{cont}, \frac{m\ C^*_{cont}+m\ \omega^*_m}{m} + C^*_{cont}\}$, that is $2\ C^*_{cont} + \omega^*_m$. If $\omega^*_m \geq 2\ C^*_{cont}$, we obtain directly a guarantee of 2.

When $\omega^*_m \leq 2\ C^*_{cont}$, as $2\ C^*_{cont}$ is greater than ω^*_m, by construction, the allotment chosen for all the tasks is lower than the allotment in the optimal malleable schedule. The work of all the tasks is lower than $m\omega^*_m$. Using the same analysis, we obtain $\omega_m \leq \frac{m\ \omega^*_m}{m} + C^*_{cont} \leq 2\ \omega^*_m$

Thus Algorithm 1 has the worst case behavior bounded by 2.

4 An Improved Algorithm with Better Average Behavior

The 2-approximation presented in [5] is based on a clever reduction of MT scheduling to the strip-packing problem, using the earliest result of [9]. It is worth stressing that the algorithm of Ludwig [5], on average behaves similarly to its worst case behavior.

Algorithm 1 has also a worst case bound equal to 2. In average it will behave similarly, most often approaching this bound. For this reason we propose a slightly more sophisticated algorithm. Its main idea for refinement is to pack more cautiously small tasks (requiring one processor only) and to use several steps of rounding off. These changes allow an improved average behavior.

Algorithm 2

```
procedure Algorithm2()
    Compute $C^*_{cont}$ and $\forall i, r_i$
    for $i = 1$ to $n$
        if $r_i \leq 1$ then $\tilde{r_i} := 1$
        else if $(r_i > 2)$ or $(r_i < 1.5)$ then $\tilde{r_i} = \lfloor r_i \rfloor$
        else $\tilde{r_i} = 2$ {refinement when $1.5 \leq r_i \leq 2$}
    $\tilde{m} = \sum_{i=1}^{n} (\tilde{r_i})$.
    find $k$, such that $t_k(r_k) = max_{1 \leq i \leq n}\{t_i(r_i)\}$
    while $\tilde{m} < m$
        $r_k = r_k + 1; \tilde{m} = \tilde{m} + 1$
        find $k$, such that $t_k(r_k) = max_{1 \leq i \leq n}\{t_i(r_i)\}$
     Schedule1 =
        find $k$, such that $r_k = max_{1 \leq i \leq n}\{r_i\}$
        $d = MinimumDateAvailableProcessor(r_k)$
        while $d + t_k(r_k) < max(C^*_{cont}, \sum_{i=1}^{n} t_i(1))$
            $Start(k) = d$; Scheduled($t_k$) = True
            find $k$, such that $r_k = max_{1 \leq i \leq n}\{r_i\}$
            $d = MinimumDateAvailableProcessor(r_k)$
        Algorithme2($\mathcal{T}$−Scheduled($\mathcal{T}$))
    if $\tilde{m} > m$ then
         Schedule2 =
            do
                find $k$, such that $r_k = max_{1 \leq i \leq n}\{r_i\}$
                $old_k = r_k$
                if $r_k > 1$ then $r_k = r_k - 1$
                $d = MinimumDateAvailableProcessor(r_k)$
                while $d + t_k(r_k) < max_{1 \leq i \leq n}\{t_i(r_i)\}$
                    $Start(k) = d$; Scheduled($t_k$) = True
                    find $k$, such that $r_k = max_{1 \leq i \leq n}\{r_i\}$
                   $d = MinimumDateAvailableProcessor(r_k)$
            until Scheduled($\mathcal{T}$) == $\mathcal{T}$ or $old_k == 1$
    else
         Schedule2 = $+\infty$
     Choose the better between Schedule1 and Schedule2
```

To evaluate the mean behavior of Algorithm 2 we use the following measure:

$$S_{Alg2} = \min\{\omega_m / C^*_{cont}, \omega_m / C_{area})\},$$

where: ω_m - a schedule length obtained by Algorithm 2, C^*_{cont} - an optimal schedule length of the continuous solution, $C_{area} = \sum_{i=1}^{n} t_i(1)/m$ - a schedule length for the uniprocessor allocation for all the tasks. Clearly, the maximum of the two values C^*_{cont} and C_{area} is the lower bound on the optimal schedule length ω^*_m for malleable tasks (discrete case), thus, S_{Alg2} indicates properly a behavior of Algorithm 2.

5 Experiments

To test mean behavior of Algorithm 2, experiments have been conducted as follows. Task processing times $t_i(1)$ have been generated from a uniform distribution in interval [1..100]. Processing speed functions is $f_i(r) = r^{1/a}, a \geq 1$. Values of parameter a have been generated from a uniform distribution in interval [1..10]. The results of the experiments are gathered in Tables 1 through 2. Each entry in these tables is a mean value for 10 instances randomly generated.

	a - different for each task			$a = 4$		
Processors	ω_m/C^*_{cont}	ω_m/C_{area}	S_{alg2}	ω_m/C^*_{cont}	ω_m/C_{area}	S_{alg2}
4	7.96	1.00	1.00	8.48	1.01	1.01
8	5.16	1.02	1.02	5.22	1.02	1.02
16	3.35	1.28	1.28	3.12	1.18	1.18
32	3.26	1.54	1.54	3.11	1.29	1.29
64	1.93	1.48	1.48	2.98	1.32	1.32
128	1.60	1.41	1.41	1.87	1.36	1.36
256	1.21	2.12	1.21	1.30	2.03	1.30
512	1.12	3.79	1.12	1.10	3.09	1.10

Table 1 ($n = 100$) illustrates an influence of a number of processors on the average behavior of Algorithm 2. Table 2 ($m = 64$) shows the impact of the number of tasks on the performance of Algorithm 2. Figure 1 illustrates an impact on the behavior of Algorithm 2 by a number of tasks with varying speed functions.

	a - different for each task			$a = 4$		
Tasks	ω_m/C^*_{cont}	ω_m/C^*_{area}	S_{alg2}	ω_m/C^*_{cont}	ω_m/C_{area}	S_{alg2}
20	1.09	4.26	1.09	1.07	3.19	1.07
40	1.11	2.60	1.11	1.33	3.05	1.33
60	1.19	1.62	1.19	1.15	1.83	1.15
80	1.17	1.68	1.17	1.28	1.52	1.28
100	1.51	1.47	1.47	1.30	1.42	1.30
120	1.48	1.41	1.41	1.43	1.18	1.18
140	1.61	1.40	1.40	1.45	1.19	1.19
160	1.78	1.25	1.25	1.57	1.18	1.18
180	2.23	1.26	1.26	1.85	1.09	1.09
200	2.05	1.10	1.10	6.01	1.00	1.00

From the experiments conducted we see that the mean behavior of the algorithm (as obtained in the above computational experiments) does not exceed value $1,54$ of the assumed lower bound of the optimal schedule length for the discrete case. The experiments show that when a number of tasks greatly exceeds a number of processors, the optimal continuous solution does not approximate well the discrete malleable one.. On the other hand, for a number of tasks being

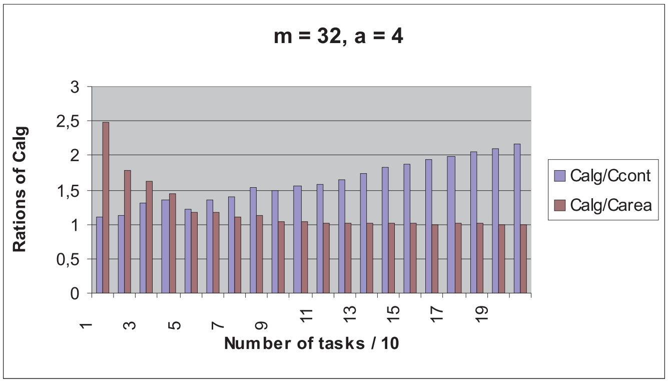

Fig. 1. Impact of a number of tasks on behavior of Algorithm 2

close to a number of processors, the continuous solution may be a good starting point for a construction of an optimal malleable schedule. Since the first is constructed in polynomial time, the second (of a good quality) may be also constructed in a short time.

6 Perspectives

Further investigations could take into account a construction of the proposed algorithm with a better worst case performance guarantee, as well as an analysis of some special (but practically important) cases, involving few parallel tasks in the system only, each requiring many processors at the same time.

References

1. J. Błażewicz, M. Drabowski, J. Węglarz: Scheduling multiprocessor tasks to minimize schedule length, *IEEE Transactions on Computers* **35**, 1986, 389–393. 191
2. J. Du, J. Y.-T. Leung: Complexity of scheduling parallel tasks systems. *SIAM Journal on Discrete Mathematics* **2**, 1989, 473–487. 191
3. R. L. Graham: Bounds for certain multiprocessing anomalies, *Bell System Tech. J.*, **45**, 1966, 1563-1581. 193
4. K. Jansen, L. Porkolab: Linear-Time Approximation Schemes for Scheduling Malleable Parallel Tasks, In *Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (soda99)*, ACM-SIAM, 1999, 490–498. 191
5. W. T. Ludwig: *Algorithms for scheduling malleable and non-malleable parallel tasks*, PhD thesis, University of Wisconsin - Madison, Department of Computer Sciences, 1995. 191, 193
6. G. Mounié, C. Rapine, D. Trystram: Efficient approximation algorithms for scheduling malleable tasks, In *Eleventh ACM Symposium on Parallel Algorithms and Architectures (SPAA'99)*, ACM, 1999, 23–32. 191

7. G. N. S. Prasanna, B. R. Musicus: The optimal control approach to generalized multiprocessor scheduling, *Algorithmica*, 1995. 191
8. A. Steinberg: A Strip-Packing Algorithm with Absolute Performance Bound 2, *SIAM Journal on Computing* **26 (2)**, 1997, 401–409.
9. J. Turek, J. Wolf, P. Yu: Approximate algorithms for scheduling parallelizable tasks, In *4th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1992, 323–332. 193
10. J. Węglarz: Modelling and control of dynamic resource allocation project scheduling systems, In S. G. Tzafestas (ed.), *Optimization and Control of Dynamic Operational Research Models*,North-Holland, Amsterdam, 1982. 192