Brief Contributions

Preemptable Malleable Task Scheduling Problem

Jacek Blazewicz, *Senior Member*, *IEEE*, Mikhail Y. Kovalyov, Maciej Machowiak, Denis Trystram, and Jan Weglarz

Abstract—The problem of optimal scheduling n independent malleable tasks in a parallel processor system is studied. It is assumed that an execution of any task can be preempted and the number of processors allocated to the same task can change during its execution. We present a rectangle packing algorithm, which converts an optimal solution for the relaxed problem, in which the number of processors allocated to a task is not required to be integer, into an optimal solution for the original problem in $O(n)$ time.

Index Terms—Scheduling, resource allocation, parallel computing.

1 Introduction

We study the following multiprocessor task scheduling problem. There are n independent and available at time zero *malleable tasks* to be scheduled for execution on m, $m \geq n$, *parallel identical processors*. At each time instant, any number of processors can be used to execute a task. However, no processor can handle more than one task at a time and the total number of processors executing the tasks should not exceed m at any time.

An amount $p_j > 0$ of work is associated with each task j. If r processors are used to execute task j in a time interval of length t, then the amount of work done on this task within this interval is equal to $f_j(r) \cdot t$, where $f_j(r) \geq 0$ is a nondecreasing *processing speed function* defined for $r \in \{0, 1, \dots, m\}$, $f_j(0) = 0$. The total amount of work done on task j must be equal to p_j, $j = 1, \dots, n$.

For each task, a schedule specifies the time intervals within which this task is executed and the numbers of processors allocated to the task within these intervals. The objective is to find a schedule that satisfies the above constraints and such that the maximum task completion time, that is, the makespan, denoted as $C_{\max}$, is minimized. Let $C^*_{\max}$ be the minimum $C_{\max}$ value.

This problem is motivated by the optimal scheduling of multiprocessor systems enabling an execution of large-scale parallel computations. A similar situation appears in scheduling multiple batch tasks. Batch scheduling consists of executing a series of independent parallel tasks using algorithms like the algorithm described in this paper. It is a crucial problem for managing the resources of clusters of PCs or workstations [5]. Today, only very simple algorithms are used (First-Come First-Served) and there is a real challenge to design efficient algorithms. Some solutions have been proposed under an assumption that processor allocation cannot change during the task execution (moldable tasks) or the number of processors is given for each task (rigid tasks) [6], but not for pure malleable tasks [8]. The existing algorithms like FCFS provide no performance guarantee. More results concerning multiprocessor tasks scheduling problems can be found in [1], [3], [7].

Two practical examples in the area of molecular dynamics and operational oceanography, which directly lead to the model considered here, are given in an earlier paper by the authors [4].

Note that parallel processors can be viewed as a discrete renewable limited resource that should be allocated to the tasks and that can speed up their execution. In the sequel, we refer to our original problem as *problem P-DSCR (discrete)*. A relaxation of this problem is *problem P-CNTN (continuous)*, in which the processor allocation is not required to be integer and the processors can be viewed as a continuously divisible renewable limited resource. In problem P-CNTN, processing speed functions $f_j(r)$ are assumed to be interpolated by linear functions between the integer points.

In problems P-DSCR and P-CNTN, time is assumed to be continuously divisible, any task can be preempted at any time, and the number of processors allocated to this task can change during its execution. We assume that there is no cost for preemption or change of processor allocation. Our algorithm for problem P-DSCR given below constructs a schedule with at most two preemptions for each task and the number of processors allocated to a task can differ between at most two values r and $r + 1$ for some r. Therefore, if there are costs for preemption or change in processor allocation, their total value can be expected to be low.

The following results were obtained in our earlier paper [4]: If all processing speed functions are convex, an $O(n)$ algorithm was presented to solve both problems P-CNTN and P-DSCR. If the functions are all concave, problem P-CNTN was shown to be solvable in $O(n \max\{m, n \log^2 m\})$ time and it was proven that the minimum makespan values for problems P-DSCR and P-CNTN coincide. Concave processing speed functions are more adequate for the majority of real-life large-scale parallel computations because the efficiency of task processing degrades while the number of allocated processors increases due to communication delays. Concave processing speeds correspond to the realistic hypothesis of parallelizing actual large numerical codes.

The main result of this paper is a *rectangle packing algorithm*, which, in the case of concave processing speed functions, converts an optimal solution for problem P-CNTN into an optimal solution for problem P-DSCR in $O(n)$ time.

2 A Rectangle Packing Algorithm for Problem P-DSCR

Denote the optimal makespan value of problem P-CNTN as $C^0_{\max}$.

Following Weglarz [9] (cf. also [2]), to explain the main idea of finding a solution for the P-CNTN problem, we introduce set

$$R = \left\{ \mathbf{r} = (r_1, \dots, r_n) \mid r_j \geq 0, \ \sum_{j=1}^{n} r_j \leq m \right\}$$

of *feasible* (with respect to problem P-CNTN) *resource allocations* and set

$$U = \{ \mathbf{u} = (u_1, \dots, u_n) \mid u_j = f_j(r_j), \\ j = 1, \dots, n, \ r \in R \}$$

of *feasible transformed resource allocations*. Denote $\mathbf{p} = (p_1, \dots, p_n)$.

- *J. Blazewicz, M. Machowiak, and J. Weglarz are with the Institute of Computing Science, Poznan University of Technology, Poland, and the Institute of Bioorganic Chemistry, Polish Academy of Sciences, Poznan, Poland. E-mail: {jblazewicz, maciej.machowiak, jan.weglarz}@cs.put.poznan.pl.*
- *M.Y. Kovalyov is with the Faculty of Economics, Belarus State University, Minsk, Belarus. E-mail: koval@newman.bas-net.by.*
- *D. Trystram is with Laboratory Informatique et Distribution, IMAG, Grenoble, France. E-mail: Denis.Trystram@imag.fr.*

Manuscript received 29 Sept. 2003; revised 23 May 2005; accepted 27 Sept. 2005; published online 22 Feb. 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0166-0903.

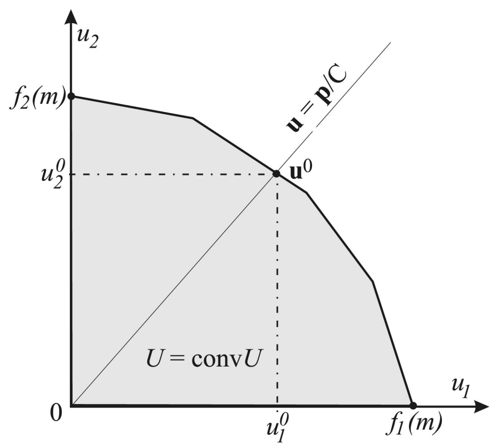

Fig. 1. An illustration for Theorem 1: straight line $\mathbf{u} = \mathbf{p}/C$ intersects a boundary of set $\mathrm{conv}U$.

Theorem 1 (Weglarz [9]). *Let $n \leq m$, $\mathrm{conv}U$ be the convex hull of the set U, i.e., the set of all convex combinations of the elements of U, and $\mathbf{u} = \mathbf{p}/C$ be a straight line in the space of transformed resource allocations given by the parametric equations $u_j = p_j/C$, $j = 1, \ldots, n$. Then, the minimum makespan value for problem P-CNTN can be found from*

$$C^0_{\max} = \min\{C \mid C > 0,\ \mathbf{p}/C \in \mathrm{conv}U\}. \quad (1)$$

From Theorem 1, it follows that the minimum makespan value $C^0_{\max}$ for problem P-CNTN is determined by the intersection point of the straight line $\mathbf{u} = \mathbf{p}/C$, $C > 0$, and the boundary of the set $\mathrm{conv}U$ in the n-dimensional space of transformed resource allocations. Denote such an intersection point by $\mathbf{u}^0$ (cf. Fig. 1).

Since functions f_j are all concave and piecewise linear, set U is a convex polytope in the n-dimensional space of transformed resource allocations. Therefore, $\mathrm{conv}U = U$ in this case. To find intersection point $\mathbf{u}^0$, we used a bisection search procedure and, from $f_j(r^0_j) = u^0_j, j = 1, \ldots, n$, we calculated optimal resource allocation $\mathbf{r}^0$ such that

$$r^0_i \geq 0,\ i = 1, \ldots, n,\ \sum_{i=1}^{n} r^0_i = m, \quad (2)$$

and

$$C^0_{\max} = p_i / f_i(r^0_i),\ i = 1, \ldots, n. \quad (3)$$

In the optimal solution for problem P-CNTN with the makespan value $C^0_{\max}$, there is a single processing interval $[0, C^0_{\max}]$ for each task and vector $\mathbf{r}^0 = (r^0_1, \ldots, r^0_n)$ represents the processor allocations for the tasks.

More complex explanations and details of the algorithm may be found in the earlier paper [4], where it was shown that $C^*_{\max} = C^0_{\max}$. Moreover, an $O(n \max\{m, n \log^2 m\})$ time algorithm to find $\mathbf{r}^0$ if the processing speed functions are all concave was presented there.

Now, we describe an algorithm which converts an optimal solution for problem P-CNTN satisfying (2) and (3) into a solution for the original problem P-DSCR with the same makespan value $C^0_{\max}$ in $O(n)$ time. Therefore, it constructs an optimal solution for problem P-DSCR in case all processing speed functions are concave.

Our algorithm, denoted as PACK, uses a geometrical interpretation of a solution for problem P-DSCR. Its parameters are obtained by rounding the parameters of an optimal solution for the corresponding problem P-CNTN.

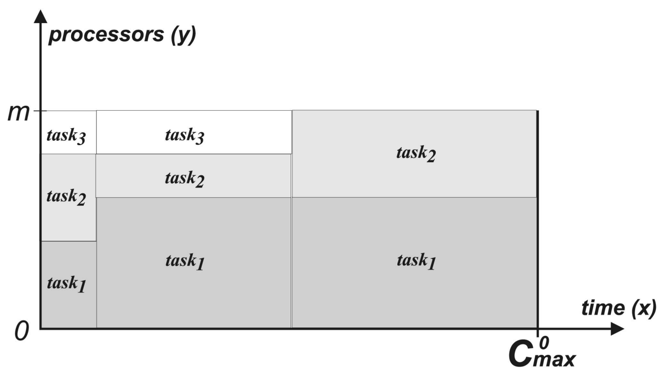

Fig. 2. Solution to problem P-DSCR.

Observe that a solution for problem P-DSCR can be represented as a collection of disjoint rectangles in the two-dimensional space (x, y), see Fig. 2.

Here, $(0, y)$ is the axis corresponding to the number of processors allocated to a task and $(0, x)$ is the time axis. Each rectangle is associated with a task. There can be several rectangles associated with the same task. Projection of a rectangle on axis $(0, x)$ determines the time interval, where the corresponding task is executed and the height of a rectangle (its length with respect to the axis $(0, y)$) determines the number of processors allocated to the task within this time interval.

It is easy to see that a collection of disjoint rectangles corresponds to an optimal solution for problem P-DSCR if and only if the following conditions are satisfied:

1. The heights of all rectangles are integer.
2. All rectangles fit into the rectangle, called BIG, with height m and width (length with respect to the axis $(0, x)$) $C^0_{\max}$.
3. Projections on axis $(0, x)$ of the rectangles corresponding to the same task may intersect only at the endpoints.
4. Let $h_1, \ldots, h_k$ and $w_1, \ldots, w_k$ be the heights and widths, respectively, of all rectangles associated with task i. Then, equation $\sum_{l=1}^{k} w_l f_i(h_l) = p_i$ must be satisfied for all i.

Condition 1 ensures that the processor allocation is integer. Condition 2 ensures that all the tasks use no more than m processors at any time and that the makespan is minimum. Condition 3 ensures that, at any time, the height of the rectangle is the number of *all* processors allocated to the corresponding task at this time. Condition 4 ensures that all the work associated with task i has been done.

We now pass to a description of the main characteristics of algorithm PACK. At the beginning, the set of tasks is partitioned into two subsets of so-called *0-tasks* and *2-tasks*. Here, task i is a 0-task if r^0_i is integer. Otherwise, it is a 2-task.

Algorithm PACK consists of two phases.

In the first phase, one rectangle is constructed for each 0-task i. It has height r^0_i and width $C^0_{\max}$. Thus, in an optimal solution for the original problem P-DSCR, each 0-task is executed on r^0_i processors in the interval $[0, C^0_{\max}]$. It means that, in the P-CNTN problem, an integer number of processors is allocated to each 0-task and its processing time is exactly equal to $C^0_{\max}$. Therefore, we may reduce P-DSCR problem by removing all such tasks.

For each rectangle of a 0-task, conditions 1, 2, and 3 are evidently satisfied. As for condition 4, since only one rectangle of width $C^0_{\max}$ and height r^0_i is associated with 0-task i, we have

$$C^0_{\max} f_i(r^0_i) = p_i.$$

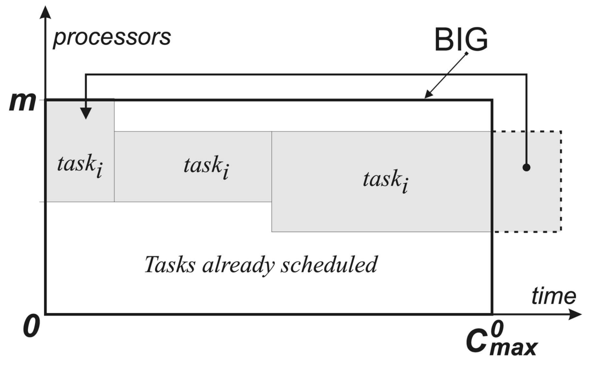

Fig. 3. Assigning a 2-task.

Rectangles for 0-tasks are packed into the rectangle BIG at its bottom and problem P-DSCR is reduced by removing 0-tasks from the original set of tasks.

In the second phase, the reduced problem is considered in which the number of tasks is $\bar{n} = n - |N_0|$ and the number of processors is $\bar{m} = m - \sum_{i \in N_0} r_i^0$, where N_0 is the set of 0-tasks.

Two rectangles with dimensions (height, width) (a_i, v_i) and (b_i, w_i) are constructed for each 2-task i. We set

$$a_i = \lfloor r_i^0 \rfloor,\ b_i = \lceil r_i^0 \rceil,\ v_i = (b_i - r_i^0)p_i/f_i(r_i^0)$$

and

$$w_i = (r_i^0 - a_i)p_i/f_i(r_i^0).$$

Observe that $b_i - a_i = 1$ for any 2-task i.

If $a_i = 0,$, then there is no corresponding rectangle. However, for convenience, we assume that this *degenerated rectangle* with zero height is present. It is clear that condition 1 is satisfied for any rectangle of a 2-task.

An idea of a construction of an optimal allocation of an integer number of processors to any 2-task i rests on rounding the processor allocation r_i^0 up and down to integer values and on representing r_i^0 as a linear combination of these both values. Next, one can easily construct two (or one, if one of the integers is equal to 0) rectangles with a total area equal to the area of the original task in the P-CNTN problem solution.

The following property of the two rectangles of the same 2-task i is used in the second phase of algorithm PACK:

Lemma 1. *The total width of two rectangles of the same 2-task is equal to* $C^0_{\max}$.

Proof. We have

$$v_i + w_i = p_i(b_i - a_i)/f_i(r_i^0) = p_i/f_i(r_i^0) = C^0_{\max}$$

for any 2-task i. □

The second phase of algorithm PACK can be described as follows: It consists of $\bar{n}$ iterations. In each iteration, rectangles corresponding to an arbitrary 2-task are assigned to the unoccupied area of the rectangle BIG and this task is excluded from further considerations. The higher of the two rectangles is assigned first. The common rule for assigning a rectangle can be called as the *rule of the southwest corner* and it is as follows: *"Assign a rectangle to the bottommost line of the unoccupied area of BIG and to the leftmost position of this line"*.

Below, we prove that a higher rectangle of a 2-task i cannot go above the top of the rectangle BIG. If it goes on the right of the right borderline of BIG, then we cut it by this borderline into two rectangles. The left of these two rectangles is considered as having

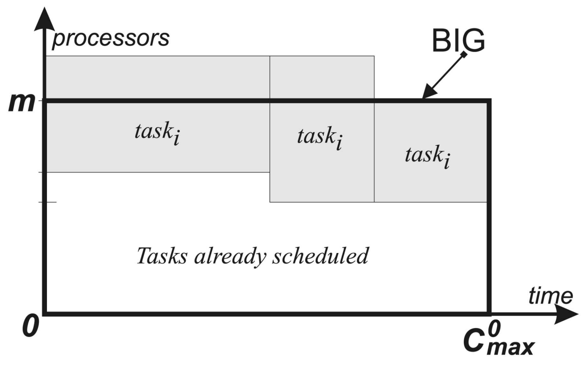

Fig. 4. The higher rectangle of a 2-task goes above the top of the rectangle BIG.

been assigned and the right rectangle is assigned according to the rule of the southwest corner. From Lemma 1, we know that the projections of the latter two rectangles on axis $(0, x)$ can intersect only at the endpoints. If they do not intersect, then, due to the facts that $b_i - a_i = 1$ and $v_i + w_i = C^0_{\max}$, the lower rectangle of the same 2-task i fills the gap between the two pieces of the higher rectangle, see Fig. 3.

Thus, in any iteration of algorithm PACK, the occupied area of the rectangle BIG looks like a staircase with at most one step. Furthermore, if there is a step, then its height is equal to 1.

Algorithm PACK stops when the rectangles corresponding to all 2-tasks are assigned.

The following three lemmas prove the validity of algorithm PACK.

Lemma 2. *The total area of the rectangles is equal to* $mC^0_{\max}$.

Proof. The total area of the rectangles for 2-tasks can be calculated as follows:

$$\sum_{i=1}^{\bar{n}}(a_i v_i + b_i w_i) = \sum_{i=1}^{\bar{n}} p_i[(b_i - r_i^0)a_i + (r_i^0 - a_i)b_i]/f_i(r_i^0)$$
$$= \sum_{i=1}^{\bar{n}} p_i r_i^0/f_i(r_i^0) = C^0_{\max}\sum_{i=1}^{\bar{n}} r_i^0 = \bar{m}C^0_{\max}.$$

To derive these equations, we used (2) and (3). Then, the total area of all the rectangles is equal to $mC^0_{\max}$. □

Lemma 3. *No rectangle for a 2-task can go above the top of the rectangle BIG.*

Proof. Assume that a rectangle of a 2-task i goes above the top of the rectangle BIG. There are two cases to consider.

In first case (no step), the higher rectangle starts from the leftmost position; in the second case (as shown in Fig. 4), there is a step of height 1 and the higher rectangle starts from this step.

In both cases, we use Lemma 1 and equation $b_i - a_i = 1$ and see that the total area occupied by the rectangles of task i is greater than the unpacked area of the rectangle BIG before task i is considered, which contradicts Lemma 2. □

Finally, we prove:

Lemma 4. *The total work on any 2-task i is equal to p_i.*

Proof. In the formulation of problem P-CNTN, we assumed that $f_i(r)$ is a linear function for $r \in [a_i, b_i]$. Since $b_i - a_i = 1,$, it can be represented as

$$f_i(r) = f_i(a_i)(b_i - r) + f_i(b_i)(r - a_i) \text{ for } r \in [a_i, b_i].$$

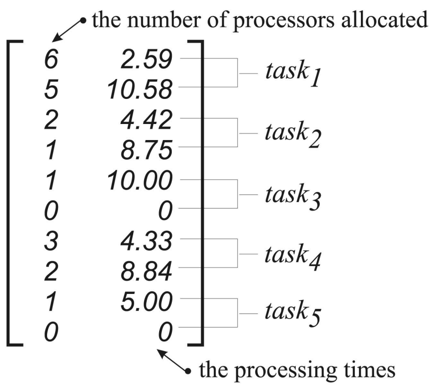

Fig. 5. Dimensions of the rectangles.

Then, the total work on a 2-task i can be calculated as

$$f_i(a_i)v_i + f_i(b_i)w_i = \frac{f_i(a_i)(b_i - r_i^0) + f_i(b_i)(r_i^0 - a_i)}{f_i(r_i^0)} p_i = p_i.$$

□

The above lemmas prove that a collection of rectangles constructed by algorithm PACK satisfies conditions 1, 2, 3, and 4. Therefore, it determines an optimal solution to problem P-DSCR if all processing speed functions are concave.

In this optimal solution, the number of processors allocated to the same task i changes at most two times between at most two different values a_i and $b_i = a_i + 1$. Furthermore, there are at most three rectangles for each task, which means that each task can be preempted at most two times.

Let us evaluate the time complexity of algorithm PACK assuming that $\mathbf{r}^0$ is given. In the first phase of this algorithm, sets of 0-tasks and 2-tasks are constructed. This operation can be performed in $O(n)$ time. Rectangles for 0-tasks can be constructed and packed into the rectangle BIG in $O(|N_0|)$ time, where N_0 is the set of 0-tasks.

In the second phase, using the rule of the southwest corner, we perform $\bar{n} = n - |N_0|$ iterations. In each iteration, at most two rectangles for some 2-task are packed into the rectangle BIG. One of these rectangles can be partitioned into two pieces. Since the occupied area of the rectangle BIG before and after each iteration looks like a staircase with at most one step, each iteration of the second phase of algorithm PACK can be implemented in a constant time. Thus, the time complexity of algorithm PACK is $O(n)$.

Example. Consider problem P-DSCR in which there are $n = 5$ tasks with processing times $p = (30, 15, 10, 20, 5)$. The total number of processors is equal to $m = 10$. The processing speed functions are the same for all tasks:

$$f_i(r) = f(r) = \sqrt{r} \text{ for } r = 1, \ldots, m, \ i = 1, \ldots, n.$$

For an optimal solution of problem P-CNTN, we have

$$C^0_{\max} = C^*_{\max} = 13.17$$
$$\mathbf{r}^0 = (5.20, 1.33, 0.76, 2.33, 0.38).$$

Using vector $\mathbf{r}^0$, determine 0-tasks and 2-tasks. Since $\mathbf{r}^0$ has no integer component, there are no 0-tasks.

Calculate $(2n \times 2)$ matrix containing dimensions of rectangles for 2-tasks, see Fig. 5.

Apply the rule of the southwest corner to pack the above rectangles into the rectangle BIG with dimensions $(m, C^0_{\max}) = (10, 13.17)$, see Fig. 6.

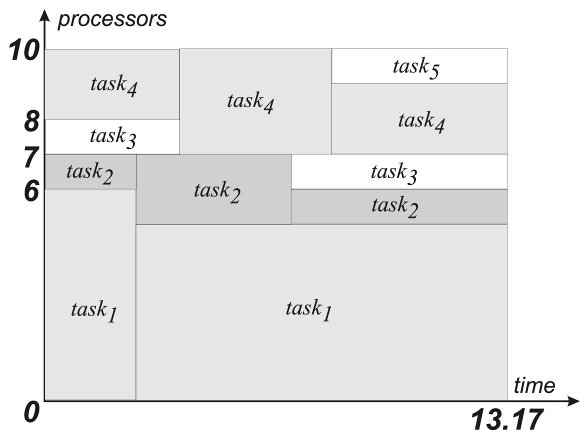

Fig. 6. Rectangles packed by algorithm PACK.

In the corresponding optimal schedule for problem P-DSCR, task 1 is executed by 6 processors in the interval $[0, 2.59]$ and by 5 processors in the interval $[2.59, 13.17]$. Processing intervals and processor allocation for the remaining tasks can be easily calculated.

3 Time Complexity Evaluation

We performed a set of computational experiments to measure the time complexity of our algorithm on average (including time spent on finding a solution for the P-CNTN problem). As we mentioned, our rectangle packing algorithm converts an optimal solution for problem P-CNTN into an optimal solution for problem P-DSCR in $O(n)$ time. Therefore, the time complexity of the algorithm solving the P-DSCR problem is determined by the complexity of finding a solution of the P-CNTN problem.

To measure an average computational time of the algorithm, we performed a set of experiments. First, the number of tasks was constant and equal to 500 and we varied m between 500 and 1,400 to see the influence of changing a number of processors on the average time complexity of the algorithm (see Table 1(a)). Next, we set a number of processors as a constant and equal to 1,000 while the number of tasks was changed between 100 and 1,000 (see Table 1(b)).

For both cases, task processing times p_i have been generated from a uniform distribution in the interval [1..100]. Processing speed functions have been chosen as $f_i(r) = r^a, 0 \le a \le 1$ and were different for each task. The results of our experiments are gathered

TABLE 1
Average Computational Time of the Algorithm for: (a) Varying Number of Processors (N = 500) and (b) Varying Number of Tasks (M = 1,000)

(a) m	n=500	(b) n	m=1000
500	0,440	100	0,039
600	0,613	200	0,151
700	0,649	300	0.275
800	0,689	400	0,533
900	0,760	500	0,798
1000	0,814	600	1,211
1100	0,930	700	1,666
1200	1,019	800	2,087
1300	1,096	900	2,150
1400	1,212	1000	2,726

in Table 1. Each entry in this table is the mean value of time (in seconds) for 25 instances, which were randomly generated according to the above description.

From the experiments conducted, we see that the average time complexity of the algorithm depends more on the number of tasks. It is justified by the fact that the time complexity of the rectangle packing algorithm PACK does not depend on the number of processors. We observe that the tested algorithm gives optimal solutions very fast for a big number of tasks and processors.

4 Concluding Remarks

Further research can be undertaken to find a more efficient algorithm for solving the relaxed problem P-CNTN. In our earlier paper [4], this problem is solved in $O(n \max\{m, nlog^2 m\})$ time. An extension to heterogeneous processors or hierarchical systems that select the grid and the problems with $n > m$ and/or arbitrary nondecreasing processing speed functions are also of interest.

Acknowledgments

This work was partially supported by the Ministry of Science and Information Society Technologies, Poland.

References

[1] J. Blazewicz, M. Drabowski, and J. Weglarz, "Scheduling Multiprocessor Tasks to Minimize Schedule Length," *IEEE Trans. Computers,* vol. 35, pp. 389-393, 1986.
[2] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, *Scheduling Computer and Manufacturing Processes.* Berlin and New York: Springer, 1996.
[3] *Handbook on Parallel and Distributed Processing,* J. Blazewicz, K. Ecker, B. Plateau, and D. Trystram, eds. Berlin and New York: Springer, 2000.
[4] J. Blazewicz, M.Y. Kovalyov, M. Machowiak, D. Trystram, and J. Weglarz, "Malleable Task Scheduling to Minimize the Makespan," *Annals of Operations Research,* vol. 129, pp. 65-80, 2004.
[5] D.E. Culler, J.P. Singh, and A. Gupta, *Parallel Computer Architecture.* Morgan Kaufman, 1999.
[6] D. Feitelson and L. Rudolph, "Parallel Jobs Scheduling: Issues and Approaches," *Lecture Notes in Computer Science,* vol. 949, pp. 1-18, 1995.
[7] W.T. Ludwig, "Algorithms for Scheduling Malleable and Nonmalleable Parallel Tasks," PhD thesis, Dept. of Computer Science, Univ. of Wisconsin-Madison, 1995.
[8] G. Mounie, C. Rapine, and D. Trystram, "A 3/2 Dual Approximation Algorithm for Scheduling Independent Monotonic Malleable Tasks," Technical Report IMAG, 2000.
[9] J. Weglarz, "Modelling and Control of Dynamic Resource Allocation Project Scheduling Systems," *Optimization and Control of Dynamic Operational ResearchModels,* S.G. Tzafestas, ed., Amsterdam: North-Holland, 1982.