A Modified Approximation Algorithm for the Small Communication Time Scheduling Problem (MAASCT)*

Vassil G. Guliashki[1]

Abstract – **The paper presents a modified approximation algorithm MAASCT, designed to solve the small communication time scheduling problem. The proposed algorithm is a modification of the recently published AASCT algorithm [2], improving its efficiency through reducing the number of computational operations, necessary in the worst case, and saving the same solution quality.[1]**

Keywords **– scheduling, makespan, communication delays**

I INTRODUCTION

The real problems for scheduling a finite number of tasks on limited number of processors require consideration of communication delays between two consecutive tasks when they are not assigned performance to one and the same processor. For convenience it is assumed that a precedence relation between two tasks i and j is available if task i needs data from task j before being started.

The paper considers the problem for making a schedule to perform n tasks on m processors, for which the task duplication is not allowed, the communication between any two processors is possible and the communication delays depend only on the corresponding tasks. The precedence constraints and the processing times are arbitrary. The objective is to find the schedule that minimizes the overall finishing time, or the "makespan". Let ρ denotes the ratio of the greatest communication delay to the smallest processing time and let the greatest communication time between any two different processors be smaller than the processing time, needed for the completion of the smallest task, i. e. $\rho \leq 1$. This problem is known as Small Communication Time problem (SCT problem).

There are surveys studying scheduling problems (see for example [1], [8], [14]), where some theoretical results about this problem are presented. As it is mentioned in [1] Picouleau has proven in 1992 that this problem is *NP*-hard. Jakoby and Reischuk have shown in [7] that the special case with unlimited number of processors and unit processing time is *NP*-hard even when the in-degree of each node is at most two. Using a similar reduction, they also proved that for a binary tree, unit processing times and arbitrary communication times the problem is *NP*-complete. For fixed $m \geq 3$, no algorithms which ensure optimal schedules are known yet. For this reason different kind of approximation algorithms have been developed (see for example [3], [5], [6], [10], [12], [13]). The parallelism of multiprocessor problem in combination with the communication delays causes difficulties at the design of approximation algorithms, because the problem is combinatorial one. The worst-case performance of all of them is as bad as possible (see [5]), especially if a great number of processors are assumed. The performance ratio for the known approximation algorithms varies around 2 and tends to 3 when the number of processors - m is fixed. The best known approximation algorithms for this problem are those presented in [10] and [5] with performance ratio $\frac{7}{3}$, and $\frac{7}{3} - \frac{4}{3m}$ correspondingly. For the problem with unlimited number of processors Hanen and Munier (see [5]) have created an approximation algorithm with $\frac{4}{3}$ performance ratio. The aim of this paper is to present the approximation algorithm MAASCT, which improves the efficiency of the algorithm AASCT [2] by means of some modifications of its steps. The new algorithm also avoids to a great extent the "anomalous behavior", arising when the number of processors increases. The computational time complexity of MAASCT algorithm is $O(\gamma n^2)$.

II PRELIMINARIES

Some symbols are introduced to define the SCT task system. The set of n tasks will be denoted by T and the corresponding processing times by $p_1, \ldots, p_n$. Let $G = (T,E)$ be a directed acyclic graph (DAG). An arc $(i,j) \in E$ corresponds to the data transfer from task i to task j, that occurs after i has been finished and before the start of j. The duration of this data transfer is a constant delay, equal to c_{ij} in case i and j are performed by different processors and 0 if i an j are performed by one and the same processor. The task system $\Im(T, p, G, c)$ is called SCT task system, if the following constraint on the communication delays holds:

$$\rho = \frac{\max_{(ij) \in E} c_{ij}}{\min_{i = 1,\ldots,n} p_i} \leq 1 \qquad (1)$$

[1] Vassil G. Guliashki is with the Institute of Information Technologies – Bulgarian Academy of Sciences, Bulgaria, Sofia 1113, "Acad. G. Bontchev" Street, Bl. 29 A, E-mail: vgoul@iinf.bas.bg, vggul@yahoo.com .

* This study is partly supported by the Ministry of Education and Science - National Science Fund, contract № I-1203 / 2002, Sofia, Bulgaria.

In some cases (see [2], [10]) the SCT system is defined by weaker conditions, but the algorithm presented in section 3 is based on condition (1).

Here is considered the problem of scheduling n tasks of the SCT task system on m processors under condition (1), where n and m are finite numbers.

A schedule $S = (t, \pi)$ assigns a starting time t_i and a processor π_i to each task i, so that

1) for any pair of tasks (i,j) if $\pi_i = \pi_j$, then $t_i + p_i \le t_j$ or $t_j + p_j \le t_i$;

2) for any arc (i,j) of G, if $\pi_i = \pi_j$, then $t_j \ge t_i + p_i$; else $t_j \ge t_i + p_i + c_{ij}$;

3) if m processors are available: $\forall\, i \in T$, $\pi_i \in \{1,\ldots,m\}$.

The makespan of the schedule is denoted by ω:

$$\omega = \max_{i \in T} (t_i + p_i) \qquad (2)$$

The optimal (minimal) makespan is denoted by ω_{opt}.

It will be assumed that the task i precedes task j if there is a path in G from i to j. The task i is called predecessor of j and the task j is called successor of i. This relation will be denoted by $i \rightarrow j$. A task i is said to be an immediate successor (resp. predecessor) of a task j if there is an arc (j,i) (resp. (i,j)) in G. For any task i we denote by $\Gamma^+(i)$ (resp. by $\Gamma^-(i)$) the set of immediate successors, (resp. predecessors) of i. In case one of immediate successors of a task j satisfies the following condition:

$$t_j < t_i + p_i + c_{ij} \qquad (3)$$

j is called the favorite successor of i. It follows from (1) that there is only one favorite successor j of i. Similarly i is called a favorite predecessor of j.

The usual approximation algorithms used for scheduling tasks on m processors, called list scheduling (LS) algorithms, build a schedule by means of a greedy process, that schedules a new task at each iteration. Assuming a partial schedule is already built for the time period $[0,t_{k-1}]$, the greedy algorithm scans each processor to find a task that is ready for it at the moment t_k and if any, to assign to it the first ready task in the list at this moment. Graham (see [3]) has proposed such algorithm for the problem without communication delays. For this case he obtained the performance ratio $\omega/\omega_{opt} = 2 - \frac{1}{m}$. Rayward-Smith has shown in [13] that any list scheduling algorithm with unit execution times and unit communication times (UET-UCT) satisfies $\omega < (3 - \frac{2}{m})\omega_{opt} - (1 - \frac{1}{m})$.

When general communication delays are considered (not necessarily SCT), an extension of the usual schema has been proposed [6], called ETF (i.e. earliest task first) that can be outlined as follows:

While there remains an unscheduled task, the set of ready tasks R (the predecessors of which have been already scheduled) is determined. Then for each couple (i, π), $i \in R$, $\pi \in \{1,\ldots,m\}$, the earliest starting time of task i on processor π, denoted by $e(i, \pi)$ is computed. Then the earliest starting time $e = \min_{(i,\pi)} e(i, \pi)$ is determined and a task i, for which there is a couple (i, π) with $e(i,\pi)=e$ is chosen and scheduled at time e. Finally a processor, for which $e(i,\pi)=e$ is assigned to i.

The ETF algorithm is analyzed in [6] and its performance ratio has the following bound: $\omega/\omega_{opt} \le 2 - \frac{1}{m} + \rho$.

As commented in [5] and [6] the relative performance of ETF can be decomposed in two parts. One of them is the Graham's bound $2 - \frac{1}{m}$ and the other is the contribution of communication delays along a path of the graph, i. e. the ratio ρ. The time complexity of ETF (see [6]) is $O(mn^2)$.

Hanen and Munier [5] proposed an approximation algorithm called FS, based on an algorithm for unlimited number of processors and on a modification of ETF algorithm. They have proved that the performance ratio of their algorithm has the following worst-case bound:

$$\omega/\omega_{opt} \le \frac{4+3\rho}{2+\rho} - \frac{2+2\rho}{m(2+\rho)} .$$

Möhring, Schäffter and Schulz [10] proposed another approximation algorithm, that is simpler than the algorithm in [5]. They first compute a schedule that regards all constraints except for the processor restrictions. This schedule is then used to construct a provable good feasible schedule for a given number of processors and as a tool in the analysis of the algorithm. The performance ratio of this algorithm is: $\omega/\omega_{opt} \le \frac{7}{3}$. In the next section is presented an approximation algorithm that in contrast to the above mentioned algorithms not is not based on a greedy procedure.

III THE MODIFIED APPROXIMATION ALGORITHM FOR THE SMALL COMMUNICATION TIME PROBLEM (MAASCT)

The algorithm MAASCT like the algorithm in [2] is based on the idea, that the arcs (i,j) of G having great c_{ij}–values should connect tasks performed by one and the same processor. In this way the tasks become favorite successors and the great delays are eliminated, which leads to reducing the greatest processing time and minimizing the makespan.

At the first step of MAASCT a consequence of tasks (chain) is constructed, beginning with the root of the spanning tree of G, so that to the current task i is added the task j for which the c_{ij} –value is maximal. In case there are many arcs having one and the same c_{ij} –value, then task j, for which p_j is maximal, is chosen as a next in the chain under composition. If there are many tasks, having one and the same processing time, then the task with smallest index is chosen. In case the current chain is composed (i.e. no more tasks can be added to it), the chain is assigned to the next processor in the list of idle processors. If there is not available idle processor, then assign the composed chain to the first processor which becomes idle. In case the starting task of the current chain needs data transfer from a task assigned to another processor, the corresponding

communication delay should be added. A new graph G' is created by removing all tasks in this chain from G. Then graph G is replaced by G' and this step is repeated until no more tasks are available for composing new chains.

At the second step the starting times for each task are calculated. At the third step are computed the finishing times for all processors. At the fourth step the processor with greatest finishing time (equal to the makespan) is determined. At the fifth step an attempt is made to rearrange the tasks on each processor in order to reduce the finishing times and the makespan. At the sixth step the replacing of groups of tasks on different processors is checked in order to reduce the makespan. At the seventh step an attempt is made to rearrange the places of tasks on the processor with greatest finishing time, trying to assign the last task on it to another processor and to reduce the makespan. Almost at each step $O(n)$ mathematical operations are performed. Steps 3÷5 may be repeated n times, and steps 2÷6 may be repeated γ times, where γ is a small positive integer. Hence there are necessary $O(\gamma n^2)$ mathematical operations for the performance in the worst case.

Step 5 and step 6 are modifications of step 3 and step 2 in AASCT algorithm (see [2]) correspondingly. Step 1 is the same as in the AASCT algorithm.

Description of MAASCT:

Step 0. Initialize $G' \equiv G$, $T' = T$, $n' = n$, s' is an empty chain. Set $icount = 0$ and $ic = 0$.

Step 1. Chose an initial task i from T' (the root of G') and add it to the current chain s'.
For j=1,…,n'; ($j \neq i$, $i \in s'$) add the task j to s', where $c_{ij} = max_{ij \in E'} \{ c_{ij} \}$. If there are many arcs (more than one), having the same c_{ij}-values, add the task j, having greatest p_j, to s'(or if there are many tasks with the same p_j, add the first j in the list to s').
Repeat the **For** cycle until there are not available successors of the last task in the chain.
If there are idle processors available, assign the chain s' to the first processor π in the list of idle processors, otherwise assign s' to the first processor, which will become idle.
Remove all task in s' and their connecting arcs from G'. Initialize s' as an empty chain.
Repeat Step 1. until there are no more unassigned tasks .

Step 2. Compute the starting time t_i for each task i on π_j as follows: $t_i = \max(f_k + c_{ki})$, where f_k are the final times for the tasks $k \in \Gamma^-(i)$ and k not assigned to π_j.

Step 3. Compute the finishing time for each processor π_i, (i=1,…,m;).

Step 4. Find the processor π_g, having the greatest finishing time T_g after processing all tasks assigned to it.

Step 5. On each processor π_l, (l=1,…,m;) rearrange all tasks, having one and the same predecessor in such order, so that the task with the smallest starting time t_i is performed first, then the task with next greater t_j *value* and so on to the task having greatest starting time t_k. In case the makespan has not been changed during this step go to Step 6. Otherwise set $ic = ic+1$. If $ic \leq n$, go to Step 3, otherwise proceed Step 6.

Step 6. For each task i on π_g, find the task j on each processor π_k, (k=1,…,m;), such that $f_i \leq t_j$. Make an attempt to reduce the makespan replacing the corresponding subchains s''_k and s''_g after the tasks j and i. In case this attempt is successful, set $icount = icount+1$. If $icount < \gamma$ go to Step 2, otherwise go to Step7.

Step 7. Try to change the place of the task before the last on π_g, and to assign the last task to another processor, reducing the makespan.

Step 8. Stop the computations (End of MAASCT).

Theorem: The time complexity of MAASCT is $O(\gamma n^2)$, where n is the number of tasks and γ is small positive integer correspondingly.

Proof: Each from the steps 1, 2, 3, 5 and 6 requires $O(n)$ mathematical operations. Since steps 3 ÷ 5 may be repeated no more than n times, and steps 2 ÷ 6 may be executed no more than γ times, the worst case performance of MAASCT algorithm requires $O(\gamma n^2)$ mathematical operations.

IV BASIC FEATURES OF MAASCT

In [3] and [13] is mentioned that when the number of processors increases, sometimes the performance of the approximation algorithm degrades ("anomalous behavior"). This is due to the essence of greedy procedures used. At Step 1. of MAASCT algorithm (like in the AASCT algorithm [2]) the composed chains are dispatched uniformly to all processors, so that the anomalous behavior is reduced to a great extent. Step 5 and Step 6 try to reduce the makespan changing the starting time of a task on the same processor and replacing tasks on different processors correspondingly. In this way they contribute anomalous behavior to be avoided.

Another important feature of MAASCT algorithm is that it has polynomial time complexity, which is better than that one of AASCT approximate algorithm for the same class of problems. In case $\gamma \leq m$, MAASCT algorithm has better or the same time complexity as the ETF algorithm (see [6]).

The mentioned features lead to the good performance of MAASCT as it is demonstrated in the next section.

V AN ILLUSTRATIVE EXAMPLE

Here is considered the illustrative example used in [5]:

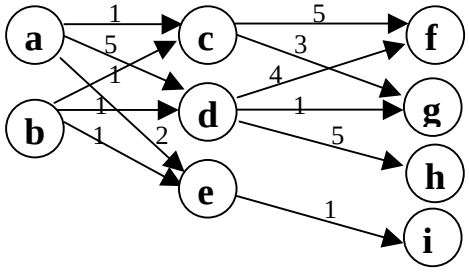

Fig. 1. Graph G with communication delays

The corresponding SCT task system for the example on Fig. 1. is presented in Table 1:

Table 1
SCT task system for the graph *G* from Fig. 1.

a	b	c	d	e	f	g	h	i
6	7	9	8	10	6	6	10	6

On Fig. 2 and Fig 3. are presented two schedules as shown in [5].

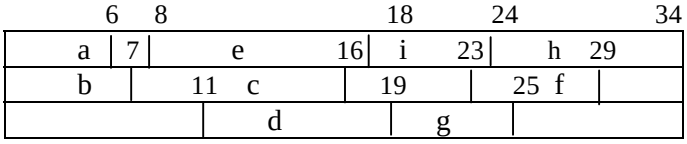

Fig. 2. An ETF schedule (3 processors)

Obviously the ETF algorithm creates a schedule with makespan (maximal finishing time) equal to 34. For the same example FS algorithm (see [5]) creates schedule with makespan equal to 29.

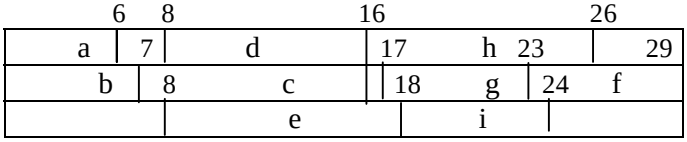

Fig. 3. A FS and MAASCT schedule (3 processors)

The MAASCT algorithm obtains the same result as the FS algorithm (see Fig. 3). Taking into account that each task can be started after it receives the necessary data from all its predecessors, the algorithm AASCT [2] obtains the same result. After Step 1 MAASCT schedules on first processor tasks *a*, *d* and *h*; on second processor – tasks *b*, *c*, *f* and *g*; and on the third processor – tasks *e* and *i*, starting *e* at moment t=8 (Step 2). The makespan is equal to 32 on the second processor (Step 4). After Step 5 MAASCT obtains the result on Fig. 3 with makespan equal to 29. At Step 6 and Step 7 the algorithm can not find new order of tasks, reducing the makespan.

On Fig. 4 is presented the result obtained by MAASCT algorithm for the same example but on two processors. After Step 1 the task schedule for the first processor is *a*, *d*, *h*, *g*; and for the second processor: *b*, *c*, *f*, *e*, *i*; The makespan is 42. After Step 6 tasks *c* and *f* are replaced by tasks *e* and *i*. The makespan is equal to 39. After Step 7 the makespan is reduced to 38 as shown on Fig. 4.

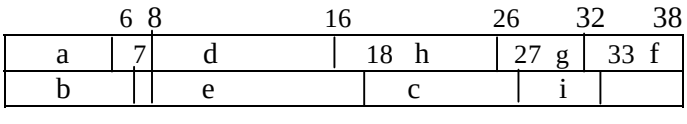

Fig. 4. A MAASCT schedule (2 processors)

VI CONCLUSIONS

An approximation algorithm called MAASCT is presented in this paper. Its time complexity is $O(\gamma n^2)$. For comparison the algorithm AASCT (see [2]) has $O(\gamma n^2(n\text{-}m))$ and ETF (see [6]) has $O(mn^2)$ time complexity. In case $\gamma \leq m$, MAASCT algorithm has better or the same time complexity as the ETF algorithm. It has also a better efficiency than the AASCT algorithm [2]. The known approximate algorithms in the literature have polynomial time complexity, but some of them have "anomalous behavior" (see [3], [13]) due the use of greedy heuristics. The algorithms AASCT [2] and MAASCT dispatch the tasks uniformly to all processors, so that the anomalous behavior is reduced to a great extent. They don't use artificial delays (see [5]). It is expected that the MAASCT algorithm will have better performance in comparison to ETF algorithm, as well as to some other approximation algorithms, based on greedy procedures.

REFERENCES

[1] Chrétienne P., C. Picouleau (1995) „*Scheduling Theory and its Applications*", P. Chrétienne, E. G. Coffman, J. K. Lenstra and Z. Liu (Eds.), 1995, John Wiley & Sons Ltd, pp. 65-90.

[2] Guliashki V., (2003) "An approximation algorithm for scheduling problem on a finite number of processors with communications delays", Proceedings of the XXXVIII International Scientific Conference ICEST'2003, held in Sofia, Bulgaria, pp. 333-336.

[3] Graham R. L. (1969) „Bounds on multiprocessing timing anomalies", *SIAM J. Appl. Math.*, Vol. 17, No. 2, pp. 416-429.

[4] Hanen C., A. Munier (1994) „Performance of Coffman-Graham schedules in presence of unit communication delays", http://citeseer.nj.nec.com/hanen94performance.html

[5] Hanen C., A. Munier (1995) „An approximation algorithm for scheduling dependent tasks on *m* processors with small communication delays", Preprint, Laboratoire Informatique Théoretique et Programmation, Institute Blaise Pascal, Université Pierre et Marie, Curie.

[6] Hwang J. J., Y. C. Chow, F. D. Anger, and C. Y. Lee (1989) „Scheduling precedence graphs in systems with interprocessor communication times", *SIAM J. Comput.*, Vol. 18, No.2, pp.244-257.

[7] Jakoby A., R. Reischuk (1992) „The Complexity of Scheduling Problems with Communication Delays for Trees", *Lecture Notes in Computer Sciences*, No. 621, Vol. 3, pp. 165-177 Springer, Berlin.

[8] Liu Z. (1995) „Worst-case analysis of scheduling heuristics of parallel systems", N° 2710, Institut National de Recherche en Informatique et en Automatique, http://citeseer.nj.nec.com/cache/papers/cs/1573/ftp:zSzzSzftp.inria.frzSzINRIAzSzpublicationzSzpubli-ps-zzSzRRzSzRR-2710.pdf/liu95worstcase.pdf

[9] Moukrim A., A. Quilliot (1998) „Scheduling with communication delays and data routing in message passing architectures", http://ipdps.eece.unm.edu/1998/scoop/moukrim.pdf

[10] Möhring R., M. Schäffter, A. Schulz (1996) „Scheduling jobs with communication delays: using infeasible solutions for approximation",http://citeseer.nj.nec.com/cache/papers/cs/5233/http:zSzzSzwww.math.tu-berlin.dezSzcogazSzpeoplezSzformer_ members_pageszSzschaeffterzSzPaperszSzExtendedAbstract517.pdf/schedu

[11] Munier A., C. Hanen (1995) „Using duplication for scheduling unitary tasks on *m* processors with unit communication delays", http://citeseer.nj.nec.com/munier95using.html

[12] Munier A., J-C. König, (1997) „A Heuristic for a scheduling problem with communication delays", *Operations Research*, 45 (1), pp. 145-148.

[13] Rayward-Smith V. J. (1987) „UET Scheduling with unit interprocessor communication delays", *Discrete Applied Mathematics* 18, 1987, pp. 55-71.

[14] Veltman B., B. J. Lageweg and J. K. Lenstra (1990) „Multiprocessor scheduling with communication delays", *Parallel Computing* 16, pp. 173-182.