An Approximation Algorithm for Scheduling Problem on a Finite Number of Processors with Communication Delays[*]

Vassil G. Guliashki[1]

Abstract – **A polynomial approximation algorithm for scheduling a finite number of tasks on m identical processors with nonzero communication times between the processors is presented. The created algorithm is compared with other approximation algorithms on the grounds of theoretical results for their worst-case performance. The possibility for anomalous behavior is commented.**

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

I. Introduction

During the last two decades the interest in solving problems for scheduling a finite number of tasks on limited number of processors grows up rapidly. The real problems of this kind require consideration of communication delays between two consecutive tasks when they are not assigned performance to one and the same processor. For convenience we will assume that a precedence relation between two tasks i and j is available if task i needs data from task j before being started.

We will consider the problem for making a schedule to proceed 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. We will assume that the greatest communication time between any two different processors is 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 [1,7,13]), 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 [6] 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 [2,4,5,9,11,12]). 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 [4]), especially if a great number of processors is assumed. The performance ratio for the known approximation algorithms varies around 2 and tends to 3 when m – the number of processors – is fixed. The best known approximation algorithms for this problem are those presented in [9] and [4] with performance ratio 7/3, and $7/3 - 4/(3m)$, correspondingly. For the problem with unlimited number of processors Hanen and Munier (see [4]) have created an approximation algorithm with 4/3 performance ratio. The aim of this paper is to present the approximation algorithm AASCT, which could to a great extent avoid the anomalous behavior, arising when the number of processors increases, and from another side could improve the performance of approximation algorithm presented in [4]. The time complexity of AASCT is $O(\gamma n^2(n-m))$.

II. Preliminaries

First of all will be defined the SCT task system. With this aim we will introduce some symbols. The set of n tasks will be denoted by T and the corresponding processing times by $p_1, ..., p_n$. 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 and 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:

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

In some cases (see [1,9]) the SCT system is defined by weaker conditions, but the algorithm presented in section 3 is based on condition (1).

Here we consider 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 $t_i + p_i \leq t_j$ or $t_j + p_j \leq t_i$;

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

[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 .

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

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

The makespan of the schedule is denoted by ω:

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

We will denote the optimal (minimal) makespan 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:

$$t_j < t_i + p_i + c_{ij}, \tag{3}$$

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 [2]) has proposed such algorithm for the problem without communication delays. For this case he obtained the performance ratio $\omega/\omega_{\text{opt}} = 2 - 1/m$. Rayward-Smith has shown in [12] that any list scheduling algorithm with unit execution times and unit communication times (UET-UCT) satisfies $\omega < (3 - 2/m)\omega_{\text{opt}} - (1 - 1/m)$.

When general communication delays are considered (not necessarily SCT), an extension of the usual schema has been proposed [5], 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), $i \in R$, $\pi \in 1, ..., m$, 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 [5] and its performance ratio has the following bound: $\omega/\omega_{\text{opt}} \leq 2 - 2/m + \rho$.

As commented in [4] and [5] the relative performance of ETF can be decomposed in two parts. One of them is the Graham's bound $2 - 1/m$ 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 [5]) is $O(mn^2)$.

Hanen and Munier [4] 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:

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

Möhring, Schäffter and Schulz [9] proposed another approximation algorithm, that is simpler than the algorithm in [4]. 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: $\omega/\omega_{\text{opt}} \leq 7/3$. 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. An Approximation Algorithm for the Small Communication Times Problem (AASCT)

The algorithm AASCT 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 AASCT is constructed a consequence of tasks (chain), 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 task consequence on the processor π_k with greatest finishing time (makespan) is considered. This consequence is investigated, checking for each of tasks in it whether this task could be changed by one of the tasks assigned to the other processors π_i, $i = 1, ..., m-1$; $i \neq k$; in such a way that the makespan would be reduced. In case the makespan has been reduced by such a change, than this step is repeated. In case on all other processors π_i, $i = 1, ..., m-1$; $i \neq k$; there does not exist a task, which could change one of the tasks on π_k, which leads to reducing the makespan, than the algorithm passes to the third step. At this step $O(n-m)$ comparisons are performed, but the number of improvements may be arbitrary large because of combinatorial nature of the problem. For this reason we restrict the number of repetitions of this step to the small positive integer γ. Hence there are necessary $O(\gamma n^2(n-m))$ mathematical operations for the performance of this step.

At the third step a check-up is done, whether some one of the tasks on the processors π_i, $i = 1, , m-1$; $i \neq k$; could be started at an earlier moment on the same processor, so that some communication delay c_{ij} could be moved to an earlier moment and the makespan would be reduced. If the makespan is reduced in this way, repeat this step. Because there are n tasks available, the maximal number of check-ups is equal to $n-1$ and the corresponding number comparisons are $O(n^2)$.

Description of AASCT:

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

Step 1. Chose an initial task i from T' (the root of G') and add it to the current chain s'.

For $j = 1, ..., n'$; $(j \neq i, i \in s')$ find $c_{ij} = \max_{ij \in E'}\{c_{ij}\}$. and add the corresponding task j to s'. If more then one such arc has the same c_{ij}-value add the task j, having greatest p_j, 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 next processor π in the list of idle processors, otherwise assign s' to the first processor, which will become idle.

Remove all tasks 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. Set $icount = 0$. Find the processor π_k having the greatest finishing time t_k after processing all tasks assigned to it.

For each task $j' \in \pi_j$, $(j = 1, ..., m-1; j \neq k)$ check whether it is possible one of the tasks $j \in \pi_k$, to be changed by j', reducing the makespan. If it is possible, do it and set $icount = icount + 1$, otherwise proceed Step 3 without repetitions and if reducing the makespan is achieved set $icount = icount + 1$, otherwise $icount = icount$.

If $icount < \gamma$ repeat Step 2, otherwise go to Step 3.

Step 3. For each processor π_j, $(j = 1, ..., m-1; j \neq k)$, for each task j' assigned to π_j, check whether it is possible some to start it on the place of a preceding task on the same processor, so that the makespan is reduced. If it is possible, do it and repeat Step 3, otherwise stop the computations (End of AASCT).

Theorem: The time complexity of AASCT is $O(\gamma n^2(n-m))$, where m and n are the number of processors and the number of tasks correspondingly.

Proof: The most time-consuming step is Step 2, which is executed γ times in the worst case. This step requires $O(n-m)$ comparisons. Taking into account the operations of Step 3, which will be performed each time Step 2 is repeated, the running time of the AASCT algorithm is $O(\gamma n^2(n-m))$. $\square$

IV. Basic Features of AASCT

As commented in [2] and [12] the increase the number of processors sometimes degrades the performance of the approximation algorithm (“anomalous behavior”). This feature has it reason in the essence of greedy procedures used. At Step 1. of AASCT algorithm the composed chains are dispatched uniformly to all processors, so that the anomalous behavior is reduced to a great extent. Step 2 and Step 3 reduce the makespan by means of changes of tasks on different processors and changing the starting time of a task on the same processor correspondingly. I this way they also contribute anomalous behavior to be avoided. The experiments performed by means of AASCT confirm this good characteristic of the proposed algorithm.

Another important feature of AASCT algorithm is that it does not use artificial delays, waiting for a more important task (in contrast to the algorithm presented in [4]) and it is reasonable to expect that the generated best schedule would have smaller makespan (finishing time) than the algorithms, which use artificial delays. The illustrative example presented in the next section confirms this presumption. The mentioned features lead to better performance of AASCT than that one of some other approximation algorithms as it is demonstrated in the next section.

V. An Illustrative Example

We will consider the illustrative example used in [4]:

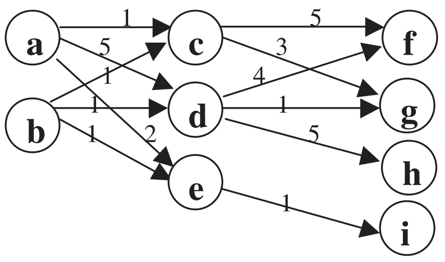

Fig. 1. Graph G with communication delays

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 [4].

Obviously the ETF algorithm creates a schedule with makespan (maximal finishing time) equal to 34. For the same example FS algorithm (see [4]) creates schedule with makespan equal to 29.

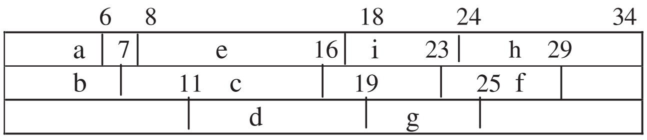

Fig. 2. An ETF schedule (3 processors)

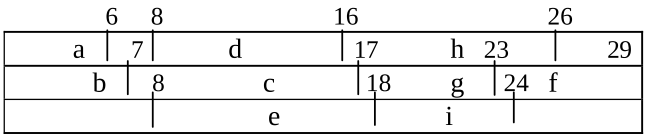

Fig. 3. A FS schedule (3 processors)

On Fig. 4 is presented the result found by means of AASCT algorithm for the same example (presented first in [4]). After Step 1 AASCT 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 $t = 8$. The makespan is equal to 28. After Step 2 AASCT obtains the result on Fig. 4 with makespan equal to 27.

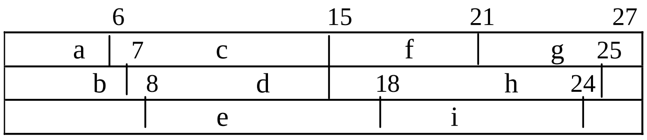

Fig. 4. An AASCT schedule (3 processors)

On Fig. 5 is presented the result obtained by AASCT 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 38. After Step 2 task a is replaced by task b and the schedule is: for the first processor is b, d, h, g; and for the second processor: a, c, f, e, i; The makespan is reduced to 37. Step 2 is repeated and task g is replaced by task e. After applying Step 3 the result on Fig. 5. is obtained. The makespan is reduced to 35.

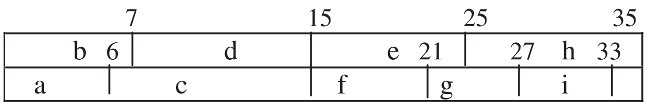

Fig. 5. An AASCT schedule (2 processors)

VI. Conclusions

An approximation algorithm called AASCT is presented in this paper. Its time complexity is $O(\gamma n^2(n - m))$. Since all approximate algorithms in the literature have polynomial computational complexity, the main criterion for comparison of their performance is the makespan (finishing time) of the generated schedule. The smaller the makespan is, the better the corresponding performance is. In this connection the relevant important features of AASCT algorithm are the avoiding anomalous behavior and artificial delays, which lead to its better performance in comparison to that one of ETF and FS algorithms, as well as (most likely) of 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] Graham R. L. (1969) "Bounds on multiprocessing timing anomalies", *SIAM J. Appl. Math.*, Vol. 17, No. 2, pp. 416-429.

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

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

[5] 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.

[6] 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.

[7] Liu Z. (1995) "Worst-case analysis of scheduling heuristics of parallel systems", No 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

[8] 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

[9] 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.dezSzcogazSzpeople zSzformer_members_pageszSzschaeffterzSzPaperszSzExtendedAbstract517.pdf/schedu

[10] 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

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

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

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