PARALELIZACIJA LOKALNOG PRETRAŽIVANJA ZA PROBLEME RASPOREDJIVANJA

PARALELLIZATION OF LOCAL SEARCH PROCEDURE FOR MULTIPROCESSOR SCHEDULING PROBLEMS[*]

Tatjana Davidović
Matematički institut SANU, Beograd, p.f. 367. email:tanjad@mi.sanu.ac.yu
Teodor Gabriel Crainic
Departement management et technologie, Université du Québec à Montréal and
Centre de recherche sur les transports, Université de Montréal
email: theo@crt.umontreal.ca

Sadržaj − *U radu se razmatra jedan od mogućih načina paralelizacije procedure lokalnog pretraživanja. Iako primenljiva na većinu problema kombinatorne optimizacije, tehnika je u ovom radu primenjena na problem rasporedjivanja u računarstvu, tj. na rasporedjivanje zadataka na izvršavanje procesorima u okviru nekog višeprocesorskog sistema. Paralelizacija je vršena na višeprocesorskom sistemu sa q+1 identičnih proce-sora imajući na umu dve strategije pretraživanja. Za prvu od njih dobijeno je skoro linearno ubrzanje, dok je kod druge paralelizacija uticala na kvalitet rešenja uvodjenjem dodatne slučajnosti.*

KLJUČNE REČI: LOKALNO PRETRAŽIVANJE, STATIČKA RASPODELA, HOMOGENI PROCESORI, KOMUNIKACIONO KAŠNJENJE.

Abstract – *Parallel neighborhood exploration strategy for parallelization of Local Search procedure is described. The main application is to Multiprocessor Scheduling Problem with Communication Delays (MSPCD), but it can be applied to other combinatorial optimization problems. The parallelization is performed on multiprocessor system containing q+1 identical processors. We tested two different search strategies and for one of them obtained almost linear speedup.*

KEY WORDS: LOCAL SARCH, STATIC SCHEDULING, HOMOGENEOUS MULTIPROCESSORS, COMMUNICATION DELAYS.

1. INTRODUCTION

A significant amount of work has already been done in implementation and analysis of various strategies for parallelization of different metaheuristics methods (see summary papers [2,9]). The general conclusions are: metaheuristics are hard for parallelization, several main ideas can be recognized, starting from the low level parallelization realized by distributing search space among processors [6,9], up to the cooperative multi-thread parallel searches [1]. In this paper we focus on parallelization of Local Search procedure which has been recognized as the main building block for varius types of metaheuristic methods [4,7]. We implemented the parallel neighbourhood exploration strategy [6,9] and applied it to Multiprocessor Scheduling Problem with Communication Delays (MSPCD).

The paper is organized as follows: in the next section combinatorial formulation of MSPCD is given. Section 3 contains the description of sequential implementation of local search procedure for solving given scheduling problem. The main ideas for parallelization of described local search procedure are given in section 4, experimental evaluations are descused in section 5, while section 6 concludes the paper.

2. PROBLEM DESCRIPTION

The Multiprocessor Scheduling Problem With Communication Delays (MSPCD) is as follows: tasks (or jobs) have to be executed on several processors; we have to find where and when each task will be executed, such that the total completion time is minimum. A duration of each task is known as well as precedence relations among tasks, i.e. what tasks should be completed before some other could begin. In addition, if dependent tasks are executed on different processors, the data transferring time (or communication delay) that are given in advance are also considered.

The tasks to be scheduled are represented by a directed acyclic graph (DAG) [3,5,8] defined by a tuple $G = (M, E, C, L)$ where $M = \{1, \ldots, n\}$ denotes the set of tasks

[*] This research has been supported in part by NSF Serbia. The first author has been supported by grant no. 1583.

(modules); $E = \{e_{ij}\ ,\ |i, j \in M\}$ represents the set of communication edges; $C = \{c_{ij}\ , e_{ij} \in E\}$ denotes the set of edge communication costs; and $L = \{l_1, \dots, l_n\}$ represents the set of task computation times (execution times, lengths). The communication cost $c_{ij} \in C$ denotes the amount of data transferred between tasks *i* and *j* if they are executed on different processors. If both tasks are scheduled to the same processor the communication cost equals zero. The set *E* defines precedence relation between tasks. A task cannot be executed unless all of its predecessors have completed their execution and all relevant data is available. Task preemption and redundant execution are not allowed.

The multiprocessor architecture A $=\{1, 2, \dots, p\}$ is assumed to contain *p* identical processors with their own local memories which communicate by exchanging messages through bidirectional links of the same capacity. This architecture is modelled by a *distance matrix* [3]. The element (*k*, *l*) of the distance matrix $D = [d_{kl}]_{p \times p}$ is equal to the minimum distance between the nodes *k* and *l*. Here, the minimum distance is calculated as the number of links along the shortest path between two nodes. It is obvious that distance matrix is symmetric with zero diagonal elements.

The scheduling of DAG G onto A consists of determining the index of the associated processor and starting time instant for each of the tasks from the task graph in such a way as to minimize some objective function. The usual objective function (that we shall use in this paper as well) is completion time of the scheduled task graph T_{max} (also referred to as makespan, response time or schedule length). The starting time of a task *i* depends on the completion times of its predecessors and the amount of time needed for transferring the data from the processors executing these predecessors to the processor that has to execute the task *i*. Depending on multiprocessor architecture the time that is spent for communication between tasks *i* and *j* can be calculated in the following way $\gamma_{ij}^{kl} = c_{ij} \cdot d_{kl} \cdot ccr$, where it is assumed that task *i* will be executed on processor *l*, task *j* on processor *k* and *ccr* represents the Communication-to-Computation-Ratio which is defined as the ratio between time for transferring the unit amount of data and the time spent for performing single computational operation. This parameter is used to describe the characteristics of multiprocessor system. In message passing systems *ccr* usually has a large value because communication links are very slow. For shared-memory multiprocessors the communication is faster since it consists of writing data from main (electronic) memory of one processor into global (also fast) memory and then into main memory of another processor. If the tasks are scheduled to the same processor, i.e. $k=l$, the amount of communication is equal to zero since $d_{kk}=0$.

3. SEQUENTIAL IMPLEMENTATION OF LOCAL SEARCH PROCEDURE

The LS procedure represents systematic search in the given neighborhood of an initial solution for the "better" solutions, the ones that result with the improvement of objective function value. Pseudo-code for this search is as follows:

```
Initialization. $x \in X$; $x'' = x$.

Repeat:
    1. $x = x''$;
    2. $(\forall x' \in N(x))$   If $f(x') < f(x'')$   then $x'' = x'$.
until $x'' = x$.
```

Here is described exhaustive search of the neighborhood, i.e. each neighbor is visited. The LS can be reduced in the sense that only some specified part(s) of neighborhood is searched. The other possible reduction is to perform First Improvement (FI) search, i.e. to stop the search when first better solution is found. If the whole neighborhood is visited searching for improvements and the best one is accepted, we will call this strategy Best Improvement (BI). Since the most consuming part of LS procedure is neighborhood exploration, our goal is to speed it up by the parallelization.

The sequential Local Search (LS) approach to MSPCD as a part of VNS procedure is proposed in [4]. The "two step" idea of constructive heuristic methods was explored in [3] for the implementation of exhaustive search over the whole solution space. Since the precedence constraints between tasks, defined by task graph, represent the partial order relation, the set of feasible permutation of tasks was used in [3] to represent the search space. Term *feasible permutation* is connected with the permutation in which order of tasks obeys precedence constraints defined by task graph. The same solution representation was used in sequential VNS implementation proposed in [4]. The solution space *S* is defined as the set of all permutations of tasks. According to the precedence relation, not all permutations are feasible. Therefore, the search space, the set of *feasible solutions* $X \subseteq S$ is defined as a set of all *feasible permutations*.

Changing the permutations, different list of tasks for scheduling are obtained, i.e. different rules for realization of first step of constructive heuristic scheduling method can be defined. For the calculation of objective function value (makespan, schedule length) which is to be minimized, we have to perform the second step of some constructive heuristic, i.e. to apply some scheduling rule. Tasks are taken one by one in order defined by selected feasible permutation, each of them is then assigned and scheduled to one of the processors. In our implementation, the Earliest start (ES) scheduling rule [3] is used.

The experiments performed with sequential implementation of permutation-based LS showed that for sparse task graphs, search space (set of feasible permutations) is too large to be exploited efficiently. We try to improve the LS procedure by its parallelization.

4. PARALELLIZATION OF LOCAL SEARCH

At the beginning of scheduling process, the initial solution has to be created. Usually, some efficient constructive heuristics are used, but sometimes, this solution is generated randomly. After initial solution is determined, the next step is its improvement by the LS procedure.

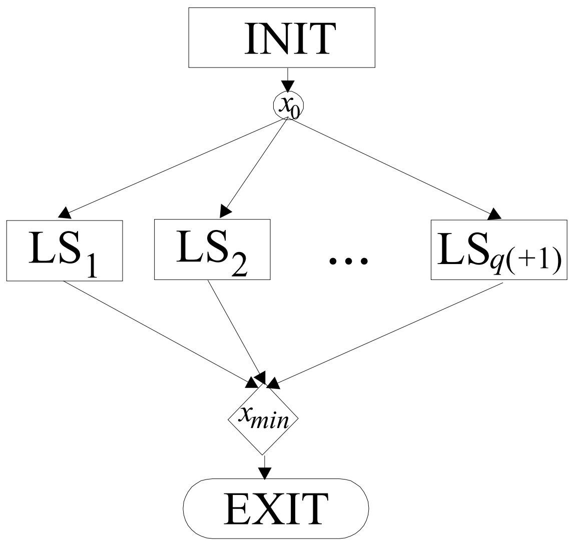

Fig. 1. Parallelization of LS procedure

The general block-diagram of parallel LS procedure is presented on Fig. 1. There are several possible ways of parallelizing the LS procedure. Here we will present the most natural one. The main idea is to divide search space into several parts and distribute these parts among processors. Each processor has to perform search process only in associated part. In general case, this strategy does not change the original sequential algorithm, and it is dedicated just to assure the speedup of search procedure. This kind of parallelization is realized by performing *parallel neighborhood exploration* (PNE) [6,9] within one iteration of LS procedure. Starting from the same current solution, each processor is exploring the associated part of the neighborhood searching for the improvement. From the computational point of view, the linear speedup should be expected since these computations are independent. Regarding the communication issues, all the improved solutions (partial local minima) should be exchanged between processors at the end of current iteration of neighborhood exploration and the best of them is propagated for further exploration.

The described PNE falls into synchronous category, since the communications are performed in strictly defined execution points. It perfectly fits with the block-diagram shown on Fig. 1. The asynchronous implementation can also be considered, but it is not the subject of this paper. The concrete implementation of this strategy depends on the target multiprocessor architecture and we will describe here the message-passing one, dedicated to the execution on distributed-memory based multiprocessor systems, such as clusters of workstations.

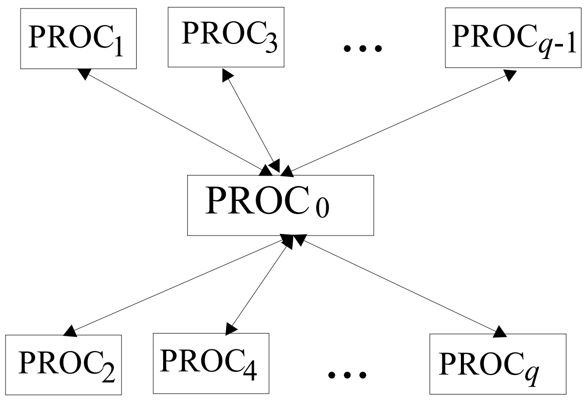

Fig. 2. Multiprocessor system for parallel LS execution

The target multiprocessor architecture is assumed to contain q+1 identical processors (with their own local memories) organized in a sort of master-slave architecture as shown on Fig. 2. Master processor is assigned communication and synchronization part of parallel execution, while slave processors perform only the calculation jobs.

At the beginning of the scheduling process, the initial solution x_0 has to be created. In current implementation of sequential VNS the initial solution (initial feasible permutation) is determined by the use of CP+ES constructive heuristic. Instead of waiting for master to determine and broadcast this initial solution, each processor creates it and therefore not only savings in communication is achieved but also idle time intervals are reduced.

Let us consider the 1-Swap neighborhood defined in [4] as changing the position of each task within its "feasible region", i.e. moving task from its original position to all others in such a way that another feasible permutation is obtained. The main idea is to divide the *n* tasks ordered in feasible permutation into several parts so that these parts can be processed simultaneously, i.e. in parallel (Fig. 3). The possible disadvantage can be quite a frequent communication, but that has to be verified experimentally.

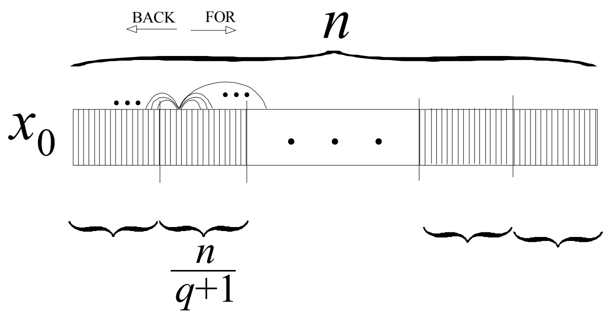

Fig. 3. Parallel 1-Swap neighborhood exploration

The search procedure consists of moving tasks in search range from their original positions to all other feasible ones (the new position of a task can be anywhere in the permutation, it is not limited by the search range, see Fig. 3.). Here, we have to define the search direction (FORward or BACKward) [4], although we can allow all neighbors of selected task to be searched. We are aware of the overlapping in search procedure, but since search is performed in parallel we will allow this case too in order to search neighborhood more efficiently and (possibly) obtain better local minimum.

The number of parts depends on the number of processors involved in parallelization of LS procedure. We consider two possible cases. The first one is when all the processors are engaged in execution of parallel LS procedure. In the second case LS is running on slave processors, while the master performs only communication and coordination operations.

For executing LS with q+1 processors in parallel, the 1-Swap neighborhood is divided among processors: each processor has to do exhaustive search over [$n/(q+1)$] succeeding tasks in the given permutation. The part of permutation searched by processor r (r=0,1,…,q is the rank (index) of the processor) is in one of the the following intervals

$$\left[r * \left\lceil \frac{n}{q+1} \right\rceil + 1,\ (r+1) * \left\lceil \frac{n}{q+1} \right\rceil \right]$$

$$\left[(q-r) * \left\lceil \frac{n}{q+1} \right\rceil + 1,\ (q-r+1) * \left\lceil \frac{n}{q+1} \right\rceil \right],$$

depending on search strategy (FI or BI). Here $\lceil x \rceil$ denotes the minimal integer grater than or equal to x. The intervals are different for the following reason: the bulk of the improvements are found in the first part (first half) of the permutation, so we need to allow master to complete its search quickly and prepare for receiving the results from slaves. On the other hand, if BI strategy is implemented the last (ending) part of permutation is assigned to master since it is the smaller one in case $n\%(q+1) \neq 0$, i.e. if q+1 does not divide n. In addition the update of objective function is quicker when last part of permutation is processed since the changed part of permutation (which is the only one to be rescheduled [4]) is smaller.

We also examined the second case, when processor 0 does not perform any calculations but since the obtained results are worse, we will not discuss this case here.

The described partition of neighborhood did not perform well, we noticed significant load imbalance of the processors. The load imbalance is the consequence of inappropriate distribution of calculations between processors. Since the required communications are almost the same for all the processors, it follows that they are computationally imbalanced. This is because the initial part of permutation requires a lot of work in search process since most of the permutation is changed and has to be rescheduled. In BI case that part is performed by the slave with the greatest rank and its load defines the total execution time of parallel LS procedure. The most consuming part in FI case is not the first one but the second. It is because the greatest probability to achieve (first) improvement is in this first part, explored by master, and the next processor (rank = 1) still has a lot of work with smaller chances to improve current solution. Therefore, we had to improve computational load balance between the processors by determining experimentaly part of the neighborhood to be processed by each processor.

In such an implementation, BI LS (denoted by BOB, Best Of Bests) is the same as in the sequential case, since in both cases all neighbors are visited and best solution is chosen. FI LS can produce different local minimum since all q(+1) FI solutions for the part of neighborhood explored are compared and the one is selected by the master. The section rule may be "best of firsts" (BOF) or "first of firsts" (FOF) and we implemented both of them.

The synchronous execution of PNE consists of performing LS procedure with specified values for parameters (FI-BI, FOR-BACK) iteration by iteration until no improvement occurs. Each processor explores its own part of search space and communicates with the master to get the next step instruction. The role of the master processor (after exploring a part of search space, if it is suppose to) is to collect all solutions from the slaves, find the best one (according to the specified rule: BOB, BOF, FOF), checks if it is better than current minimum and distribute it to the others. If the current best solution is not improved, it sends STOP message to the others and reports the best found solution to the user.

To minimize the required communication, slave processors first send only the "improved" value of the objective function. If the improvement is obtained, the slave which found that schedule is identified and asked to send the corresponding best solution, i.e. the feasible permutation of tasks, to the master so that it can be broadcasted to all the slaves for the next iteration.

5. EXPERIMENTAL EVALUATION

In this section the experiments we performed with different implementations of parallel LS procedure are described. The tests are run on SUN Enterprise 10000 multiprocessor system where 1-5 processors are used. Programs are developed in C programming language and MPI communication library is used for the communication between processors. The scheduling results are presented for 10 task graphs with the number of tasks ranging from 50 to 500 with the increment of 50, while the edge density is around 30%. Graphs are generated randomly as described in [4].

The results of BI and FI parallel local search in the whole neighborhood (in both directions FOR and BACK) with up to q+1=5 processors are given in Table 1. First column contains the number of slave processors. In the next column is given average over 10 instances of random task graphs value of number of iterations. Third column of Table 1 contains the average over 10 test examples value of scheduling length obtained by parallel LS procedure. The CPU times spent by each processor are given in the fourth column (containing q+1 values) with the computation time of each processor given in parentheses below. The last three columns contain total workload of the processors, total parallel execution time (wall clock time) and the speedup factor respectively.

The sequential execution is presented in the first row of each part of the Table 1. It is very useful to have it here for the comparisom matter. The rest of the data show the following. In the BI search, the results are the same and only benefit is in the speedup. As we can see, speedup factor is almost linear. The difference between computation time (in parentheses) and the total execution time shows that the time spent for communication and synchronization between processors is not very large, meaning that we manage to achieve both of our goals: minimizing the communications and maximizing the load balance between processors. The data about so called wall clock time, the total parallel execution time are important for the estimation of CPU time used by operating system to handle parallel execution. As we can see, this time is usually not significant.

The above discussion does not hold for the case when FI search is performed. First of all the results are not the same. Moreover, they may be different for different (FOF) executions since it is almost impossible to predict which processor will complete its computations first and report the new starting point for the next iteration. This property significantly influences the final result introducing randomness into the search which may be very useful when LS is incorporated into metaheuristic procedures. Moreover, the load balance between processors is again violated, since it may change from iteration to iteration. Consequently, the achieved speedup is smaller.

We experimented with restricted versions of parallel neighborhood exloration (FOR and BACK search) and made very similar conclusions.

6. CONCLUSION

In this paper we implemented a parallel variant of local search procedure for Multiprocessor Scheduling Problem with Communication Delays. The implementation is based on parallel neighborhood exploration and tested on multiprocessor architecture with relatively small number of processors. Two variants of the search are used and for each of them results are analysed. Best improvement search can be easily parallelized with a very good performance. Adding new processors in this case should not be the problem. The first improvement search seem to be harder for parallelization, it shows quite stochastic behaviour. Anyway, that kind of behaviour may be very useful during complex searches defined by some metaheuristic procedures.

Table 1. Scheduling results of parallel neighborhood exploration for 10 random task graphs

BI search						
q	n. it.	f_{opt}	CPU time	∑ CPU	W time	Speedup
0	4.10	2750.90	646.71 (646.71)	646.71	652.09	1.00
1	4.10	2750.90	348.58 343.10 (348.55) (295.95)	691.67	353.25	1.86
2	4.10	2750.90	237.80 241.78 237.66 (203.62) (241.60) (179.55)	717.67	424.11	2.67
3	4.10	2750.90	182.23 185.66 185.31 184.41 (131.08) (185.60) (155.46) (118.46)	744.54	188.55	3.48
4	4.10	2750.90	145.38 145.35 146.70 145.11 142.24 (126.84) (129.39) (146.22) (112.85) (49.58)	742.79	147.75	4.41

FI search						
q	n. it.	f_{opt}	CPU time	∑ CPU	W time	Speedup
0	10.70	2758.80	749.66 (749.01)	749.66	773.85	1.00
1	7.60	2778.20	325.15 325.34 (220.39) (212.02)	650.50	340.29	2.30
2	6.50	2779.20	284.23 288.69 288.39 (166.25) (217.28) (209.81)	861.31	297.51	2.60
3	6.20	2778.20	245.86 245.72 248.79 247.26 (117.68) (154.29) (203.89) (200.16)	987.63	254.99	3.01
4	6.30	2777.90	222.15 222.30 224.74 224.38 225.20 (98.27) (83.08) (155.84) (166.59) (187.99)	1118.77	230.29	3.33

REFERENCES

[1] T.G. Crainic. Parallel computation, co-operation, tabu search. In C. Rego and B. Alidaee, editors, *Adaptive Memory and Evolution: Tabu Search and Scatter Search*. Kluwer Academic Publishers, 2002.

[2] T. G. Crainic and M. Toulouse. Parallel strategies for meta-heuristics. In F. Glover and G. Kochenberger, editors, *State-of-the-art Handbook in Metaheuristics*. Kluwer Academic Publishers, 2003.

[3] T. Davidović. Exaustive list--scheduling heuristic for dense task graphs. *YUJOR*, 10(1):123--136, 2000.

[4] T. Davidović, P. Hansen, and N. Mladenović. Variable neighborhood search for multiprocessor scheduling problem with communication delays. In *Proc. MIC'2001, 4th Metaheuristic International Conference*, pages 737--741, Porto, Portugal, 2001.

[5] Y.-K. Kwok and I. Ahmad. Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. *J. Parallel and Distributed Computing*, 47:58--77, 1997.

[6] F. García-López, B. Melián-Batista, J. A. Moreno-Pérez, and J. M. Moreno-Vega. The parallel variable neighborhood search for the *p*-median problem. *Journal of heuristics*, 8(3):375--388, May 2002.

[7] S. C. Porto and C. C. Ribeiro. Parallel tabu search message-passing synchronous strategies for task scheduling under precedence constraints. *J. Heuristics*, 1(2):207--223, 1996.

[8] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. on Parallel and Distributed Systems*, 4(2):175--187, February 1993.

[9] M. G. A. Verhoeven and E. H. L. Aarts. Parallel local search. *Journal of heuristics*, 1:43--65, 1995.