Multiprocessor Scheduling with Support by Genetic Algorithms - based Learning Classifier System

Jerzy P. Nowacki[1], Grzegorz Pycka[2] and Franciszek Seredyński[1,3]

[1] Polish -Japanese Institute of Information Technologies
Koszykowa 86, 02-008 Warsaw, Poland
[2] Polish Telecom
Sw. Barbary 2, 00-686 Warsaw, Poland
[3] Institute of Computer Science, Polish Academy of Sciences
Ordona 21, 01-237 Warsaw, Poland
e-mail: jpnowacki@pjwstk.waw.pl, grzes@crt.tpsa.pl, sered@ipipan.waw.pl

Abstract. The paper proposes using genetic algorithms - based learning classifier system (CS) to solve multiprocessor scheduling problem. After initial mapping tasks of a parallel program into processors of a parallel system, the agents associated with tasks perform migration to find an allocation providing the minimal execution time of the program. Decisions concerning agents' actions are produced by the CS, upon a presentation by an agent information about its current situation. Results of experimental study of the scheduler are presented.

1 Introduction

The problem of multiprocessor scheduling formulated as designing effective algorithms to minimize the total execution time of a parallel program in a parallel architecture remains a challenge for practitioners and researchers. Recently proposed scheduling algorithms show the importance and effectiveness of such issues as *clustering* [1], *clustering* and *task-ordering* [5] or *dynamic critical-path* [3]. The increasing number of algorithms applies methodologies of natural computation, such as *mean-field annealing* [6], *genetic algorithms* [4] or *cellular automata* [7].

In the paper we propose to use a CS [2] to solve the scheduling problem. CSs constitute the most popular approach to genetic algorithms-based machine learning. We investigate a possibility to create a scheduling algoritm based on multi-agent interpretation of the scheduling problem and controlling behavior of the system with use of a single CS.

Section 2 of the paper briefly describes a concept of CSs. Section 3 discusses accepted models of parallel programs and systems, proposes a multi-agent interpretation of the scheduling problem, and presents a concept of a scheduler. Section 4 describes an architecture of a CS used for scheduling. Results of experimental study of the proposed scheduler are presented in Section 5.

2 Genetic Algorithms-based Learning Classifier System

A learning CS operating in some environment is a system which is able to learn simple syntactic rules to coordinate its actions in the environment. Three components of a CS can be pointed: CLASSIFIERS - a population of decision rules, called *classifiers*, CREDIT ASSIGNMENT - a system to evaluate rules, and DISCOVERY OF RULES - a system to modify or discover new rules.

A classifier c is a condition-action pair $c =< \textbf{condition} >:< \textbf{action} >$ with the interpretation of the following decision rule: if a current observed state of the environment matches the **condition**, then execute the **action**. The conditional part of a classifier contains some description of the environment, expressed with use of symbols $\{0,1\}$, and additionally a *don't-care* symbol #. The action part of a classifier contains an action of the CS, associated with the condition.

A usefulness of a classifier c, applied in a given situation, is measured by its *strength* str. A real-valued strength of a classifier is estimated in terms of rewards for its action obtained from the environment. If a measurement of the environment matches a conditional part of a classifier then the classifier is activated and becomes a candidate to send its action to the environment. Action selection is implemented by a competition mechanism based on *auction* [2], where the winner is a classifier with the highest strength.

To modify classifier strengths the simplified *credit assignment* algorithm [2] is used. The algorithm consists in subtracting a tax of the winning classifier from its strength, and then dividing equally the reward received after executing an action, among all classifiers matching the observed state.

A strength of a classifier has the same meaning as a *fitness function* of an individual in genetic algorithm (GA) (see, e.g. [2]). Therefore, a standard GA with three basic genetic operators: selection, crossover and mutation is applied to create new, better classifiers.

3 Multi-agent Approach to Multiprocessor Scheduling

A multiprocessor system is represented by an undirected unweighted graph $G_s = (V_s, E_s)$ called a *system graph*. V_s is the set of N_s nodes representing processors and E_s is the set of edges representing channels between processors. A parallel program is represented by a weighted directed acyclic graph $G_p =< V_p, E_p >$, called a *precedence task graph* or a *program graph*. V_p is the set of N_p nodes of the graph, representing elementary tasks. The weight b_k of the node k describes the processing time needed to execute task k on any processor of the system.

E_p is the set of edges of the precedence task graph describing the communication patterns between the tasks. The weight a_{kl}, associated with the edge (k,l), defines the communication time between the ordered pair of tasks k and l when they are located in neighboring processors. If the tasks k and l are located in processors corresponding to vertices u and v in G_s, then the communication delay between them will be defined as $a_{kl} * d(u,v)$, where $d(u,v)$ is the length of the shortest path in G_s, between u and v.

The purpose of *scheduling* is to distribute the tasks among the processors in such a way that the precedence constraints are preserved, and the *response time* T is minimized. T depends on the *allocation* of tasks in the multiprocessor topology and *scheduling policy* applied in individual processors:

$$T = f(allocation, scheduling\_policy). \tag{1}$$

We assume that the scheduling policy is fixed for a given run. The scheduling policy accepted at this work assumes that the highest priority among tasks ready to run in a given processor will have the task with the greatest number of successors. The priority p_k of a task k is calculated using the following recurrent formula:

$$p_k = s_k + \sum_{n_k=1}^{s_k} p_{k_{n_k}}, \tag{2}$$

where, s_k is the number of immediate successors of a task k, and $p_{k_{n_k}}$ is a priority of the n_k immediate successor of the task k.

For the purpose of the scheduling algorithm we specify two additional parameters of a task k mapped into a system graph: a Message Ready Time (MRT) predecessor of the task k, and the MRT successor of the k. A MRT predecessor of a task k is its predecessor which is the last one from which the task k receives data. The task can be processed only if data from all predecessors arrived. A MRT successor of the task k is a successor for which the task is the MRT predecessor.

We propose an approach to multiprocessor scheduling based on a multi-agent interpretation of the parallel program. We assume that an agent associated with a given task can perform a migration in a system graph. The purpose of migration is searching for an optimal allocation of program tasks into the processors, according to (1). We assume that decision about migration of a given agent will be taken by a CS, after presentation by the agent a local information about its location in the system graph.

4 An Architecture of a Classifier System to Support Scheduling

To adjust the CS to use it for scheduling we need to interpret the notion of an environment of the CS. The environment of the CS is represented by some information concerning a position of a given task located in a system graph. A message containing such a information will consist of 7 bits:

- bit 0: value 0 - task does not have any predecessors;
  value 1 - the task has at least one predecessor
- bit 1: value 0 - the task does not have any successors;
  value 1 - the task has at least one successor

- bit 2: value 0 - the task does not have any brothers; value 1 - the task has brothers
- bits 3 and 4:
  values 00 - none MRT successor of the task is alocated on the processor where the task is allocated; values 01 - some MRT successors are allocated on the same processsor where the task is allocated; values 11 - all MRT successors are alocated on the same processor where the task is allocated; values 10 - the task does not has any MRT successors
- bits 5 and 6:
  values 00 - none MRT predecessor of the task is alocated on the processor where the task is allocated; values 01 - some MRT predecessors are allocated on the same processsor where the task is allocated; values 11 - all MRT predecessors are alocated on the same processor where the task is allocated; values 10 - the task does not has any MRT predecessors.

The list of actions of a CS contains 8 actions:

- *action 0*: **do_nothing** - the task does not migrate from a given location (processor) to any other processor of the system
- *action 1*: **random_action** - randomly chosen action from the set of all actions, except the *action 1*, will be performed
- *action 2*: **random_node** - the task migrates to one of randomly chosen processors of the system
- *action 3*: **pred_rnd** - the task migrates to a processor where randomly selected predecessor of the task is located
- *action 4*: **succ_rnd** - the task migrates to a processor where randomly selected successor of the task is located
- *action 5*: **less_neighbours** - the task migrates to a processor where the smallest number of neighbours of the task is located
- *action 6*: **succ_MRT** - the task migrates to a processor where its MRT successor is located
- *action 7*: **pred_MRT** - the task migrates to a processor where its MRT predecessor is located.

Conditional part of a classifier contains information about specific situation of a given task which must be satisfied to execute the action of the classifier. For example, a classifier $< \#1\ \#0\ 0\ \#0\ \ >:< 6 >$ can be interpreted in the following way: **IF**: it does not matter whether the task has predecessors or not (symbol: #) **AND IF**: the task has successors (symbol: 1) **AND IF**: it does not matter whether the task has brothers or not (symbol: #) **AND IF**: none among MRT successors of the task is located on the processor where the task is located (symbols: 00) **AND IF**: none among MRT predecessors of the task is located on the processor where the task is located or the task does not has MRT predecessors (symbols: # 0) **THEN**: move the task to the processor where is located a MRT successor of the task (symbol: 6).

Fig. 1. Initial population of classifiers in few first steps of working the scheduler.

5 Experimental Results

Experiment #1: Step by step simulation (problem: $gauss18 > full2$)

We will analyze some initial steps of the work of the scheduler, solving the scheduling problem for a program graph $gauss18$ ([3], see Fig. 2a) processed in the 2-processor system $full2$. The program contains 18 tasks, and is initially allocated as shown in Fig. 2b with response time $T = 74$. Fig. 1 shows an initial population of the CS containing 20 classifiers with initial values of the strenght of each equal to 300.

The agent A_0 sends first its message $< 0100010 >$ to the CS. The message describes the actual situation of the task 0 (as shown in Fig. 2a, b) and contains the following information: the task 0 does not have any predecessor; it has successors; it does not have any brothers; all MRT successors are located on a processor different than the processor where is located the task 0, and the task does not have any MRT predecessors.

The message matches three classifiers of the CS: the classifier 0, 7 and 14. A winner of the competition between them is the classifier 14, and its action $< 3 >$ is passed to the agent A_0. The action says: migrate to a processor where your randomly chosen predecessor is located. The agent A_0 can not execute this action, because the task 0 does not have any predecessors. So, the allocation of tasks and corresponding value of T is not changed. The CS receives a reward

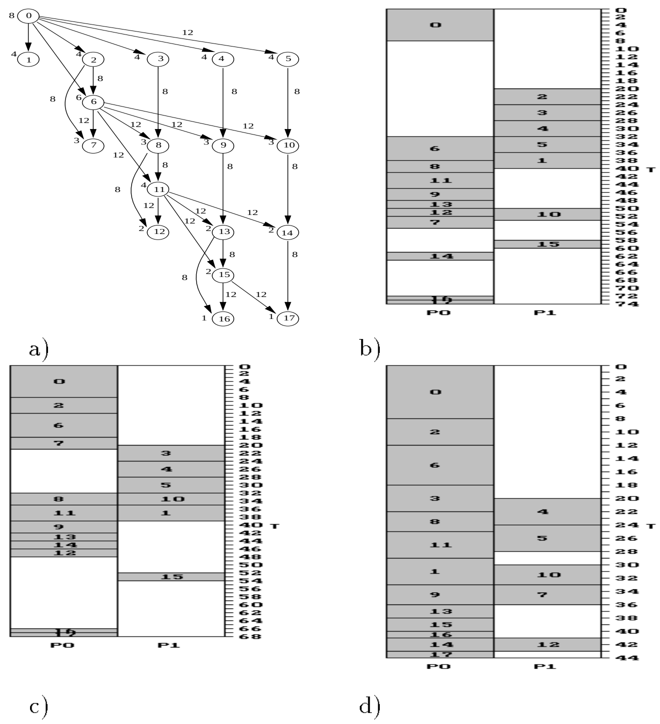

Fig. 2. Program graph *gauss*18 (a) and Gantt charts (b), (c), (d) for allocations of tasks in *full*2, corresponding to actions of classifiers shown in Fig. 1.

equal to 1 for this action, because it does not change the value of T (the value of the reward - a user defined parameter). The reward increases the strength of the classifier 14. New strengths of classifiers, as shown in Fig. 1, in the column corresponding to the agent 0, are the result of applying taxes.

The next agent which sends a message to the CS is the agent A_1, and the message is $< 1011000 >$. The message matches classifiers 2, 12 and 13. A winner of the competition is the classifier 2, which sends the action $< 1 >$ to the A_1. As the result of this action (random_action), the action 0 is chosen (do_nothing). The allocation of tasks remains the same, and the classifier 2 receives the reward equal to 1. All classifiers pay the *life* tax, the classifiers 2, 12, and 13 pay *bid* tax, also the winner pays a tax, what results in new values of strength of classifiers.

The agent A_2 sends to the CS the message $< 1110000 >$. It matches only the classifier 0 and the action $< 7 >$ is executed by the agent. The execution of the action results in the migration of the task 2 from the processor 1 to the processor 0, where the task MRT predecessor is located. Changing allocation of tasks reduces T to the value 68 (see, Fig. 2c). The classifier 0 receives the reward equal to 100 for improvement (the user defined value for improvement) of T.

Fig. 3. The best response time received for different program and system graphs.

Next, the agent A_3 sends the message $< 1110000 >$ - the same message as the one sent by the agent A_2. The message matches only the classifier 0, what causes the execution by the agent of the same action as previouly, and the migration of the task 3 to the processor 0. The new value of $T = 61$ is better than the previous value, and the classifier 0 receives again the reward equal to 100.

The agent A_4 sends the same message to the CS as agents A_3 and A_2 sent. However, an attempt to execute the action $< 7 >$ by the agent, i.e. migration of the task 4 from the processor 1 to the processor 0, increases T to the value 62, so the execution of the action is cancelled and the classifier 0 receives the reward equal to 0 (the user defined value for causing the result worse).

The action executed by the agent A_5 does not change the value of T. The message $< 1111000 >$ of the agent matches classifiers 5 and 13, and the classifier 5 with the action $< 5 >$ is the winner. The execution of the action, i.e. the migration of the task 5 from the processor 1 to the same processor, obviously does not change tasks' allocation.

Agents execute their actions sequentially, in the order of their numbering in the program graph. After the execution of an action by the agent A_{17}, the sequence of actions is repeated again starting from the agent A_0. In the considered experiment, actions of next several agents do not improve the value of T. The next improvement of T appears as the result of the execution of an action by the agent A_{15} ($T = 46$). Last migration of a task, which causes decreasing T to $T = 44$ takes place in the iteration 38. Found value of T (see, Fig. 2d) is optimal and can not be improved.

Experiment #2: Response time for different scheduling problems

The scheduling algorithm was used to find response time T for deterministic program graphs such as $tree15, gauss18, g18, g40, fft64$ known from literature, processed in different topologies of multiprocessor systems, such as $full2, full5, ring8, cube8, deBruin$. Also a number of random graphs were used, with $25, 50$ and 100 (in the average) tasks ($Rnd25\_x$, $Rnd50\_x$, $Rnd100\_x$), where x denotes ratio of the average communication time a_{kl} in a program graph to the average processing time b_k of tasks in the program graph. Fig. 3 summerizes results. Results obtained for deterministic graphs are the same as known in literature. Results obtained for random graphs were compared with results (not shown) obtained with use of GA-based algorithms, such as parallel GAs of *island* and *diffusion* models. Results obtained with use of the scheduler are significantly better than with use of parallel GAs.

6 Conclusions

We have presented results of our research on development scheduling algorithms with support of the scheduling process by genetic algorithm-based learning classifier system. Results of experimental study of the system are very promising. They show that the CS is able to develop effective rules for scheduling during its operation, and solutions found with use of the CS outperform ones obtained by applying non-learning GA-based algorithms.

Acknowledgement
The work has been partially supported by the State Committee for Scientific Research (KBN) under Grant 8 T11A 009 13.

References

1. S. Chingchit, M. Kumar and L. N. Bhuyan, A Flexible Clustering and Scheduling Scheme for Efficient Parallel Computation, in *Proc. of the IPPS/SPDP 1999*, April 12-16, 1999, San Juan, Puerto Rico, USA, pp. 500-505.
2. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989
3. Y. K. Kwok and I. Ahmad, Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors, *IEEE Trans. on Parallel and Distributed Systems.* 7, N5, May 1996, pp. 506-521.
4. S. Mounir Alaoui, O. Frieder and T. El-Ghazawi, A Parallel Genetic Algorithm for Task Mapping on Parallel Machines, in J. Rolim et al. (Eds.), *Parallel and Distributed Processing*, LNCS 1586, Springer, 1999, pp. 201-209.
5. A Radulescu, A. J. C. van Gemund and H. -X. Lin, LLB: A Fast and Effective Scheduling for Distributed-Memory Systems, in *Proc. of the IPPS/SPDP 1999*, April 12-16, 1999, San Juan, Puerto Rico, USA, pp. 525-530.
6. S. Salleh and A. Y. Zomaya, Multiprocessor Scheduling Using Mean-Field Annealing, in J. Rolim (Ed.), *Parallel and Distributed Processing*, LNCS 1388, Springer, 1998, pp. 288-296.
7. F. Seredynski, Scheduling tasks of a parallel program in two-processor systems with use of cellular automata, *Future Generation Computer Systems* 14, 1998, pp. 351-364.