Scheduling Tasks of a Parallel Program in Two-Processor Systems with use of Cellular Automata

Franciszek Seredyński *

Institute of Computer Science, Polish Academy of Sciences
Ordona 21, 01-237 Warsaw, Poland
sered@ipipan.waw.pl

Abstract. Cellular automata (CA) are proposed to design scheduling algorithms for allocating tasks of a parallel program in multiprocessor systems. For this purpose a program graph is considered as CA which control a process of migration of tasks in a multiprocessor system. In the first phase of the algorithm, effective rules for CA are discovered by a genetic algorithm. In the second phase, for any initial allocation of tasks, CA-based scheduler is able to find an allocation which minimizes the total execution time of the program in a multiprocessor system.

1 Introduction

One of current issues in parallel computing is constructing efficient algorithms for scheduling tasks of a parallel program in a multiprocessor architecture. Current works [1, 3, 5] concerning a scheduling problem are oriented either on selection problems for which *exact* solutions can be constructed or designing *heuristic* algorithms to find near-optimal solutions for more general cases. To this stream of research belong in particular scheduling algorithms based on applying techniques derived from nature such as simulated annealing, genetic algorithms or neural networks.

In this paper we propose to use for scheduling very promising but still not well enough explored technique based on applying CA. CA present a distributed system of single, locally interacting units which are able to produce a global behavior in a process of self-organization. Recent results [2] show that CA combined with evolutionary techniques can be effectively used to solve e.g. complex classification problems.

The remainder of the paper is organized as follows. The next section discusses accepted models of a parallel program and a parallel system in the context of a scheduling problem. Section 3 provides a background on CA. Section 4 contains a description of a proposed CA-based scheduling algorithm with genetic algorithm-based engine for discovering scheduling rules. Results of experimental study of the scheduler are presented in Section 5. Last section contains conclusions.

* The work has been partially supported by the State Committee for Scientific Research (KBN) under Grant 8T11A 024 11.

2 Scheduling Problem

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 of the system graph representing processors with their local memories of a parallel computer of MIMD architecture. E_s is the set of edges representing bidirectional channels between processors and define a topology of the multiprocessor system.

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, which are indivisible computational units. Weights b_k of the nodes describe the processing time needed to execute a given task on any processor of a given multiprocessor system. There exists a precedence constraint relation between the tasks k and l in the precedence task graph if the output produced by task k has to be communicated to the task l. E_p is the set of edges of the precedence task graph describing the communication pattern between the tasks. Weights a_{kl} of the edges describe a communication time between pairs of tasks k and l, when they are located in neighbor processors.

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 (the total execution time) is minimized.

For the purpose of next sections we will introduce some additional notions. We assume that for each node k of a precedence task graph there are defined sets of *predecessors(k)*, *brothers(k)* (i.e. nodes having at least one common predecessor), and *successors(k)*. We assume that we will be able to define for each node k of a precedence task graph two its parameters: *level* and *co-level*. A level h_k of a node k is defined as the maximal length of the longest path from a node k to an exit node. The co-level d_k of a node k is defined as the length of the longest path from the starting node to the node k. Values of a level and co-level of a given task are *static* and do not depend on an allocation of a program graph in processors of a parallel system. Values of a level or co-level calculated for tasks of a program graph allocated in a system graph we will call a *dynamic* level or co-level, respectively.

The response time T for a given allocation of tasks in multiprocessor topology depends on a *scheduling policy* applied in a given processor. We will assume that a scheduling policy is defined for a given run of a scheduling algorithm, and is the same for all processors of the system.

3 Cellular Automata

One dimensional CA [6] is a collection of two-states elementary automata arranged in a lattice of the length N, and locally interacted in a discrete time t. For each cell i called a central cell, a neighborhood of a radius r is defined and consisting of $n_i = 2r + 1$ cells, including the cell i.

It is assumed that a state q_i^{t+1} of a cell i at the time $t + 1$ depends only on states of its neighborhood at the time t, i.e.

$$q_i^{t+1} = f(q_i^t, q_{i1}^t, q_{i2}^t, ..., q_{ni}^t), \quad (1)$$

and a transition function f_g, called a *general rule*, which defines a rule of updating a cell i. A length L_g of a general rule and a number of neighborhood states for a binary uniform CA is $L_q = 2^n$, where $n = n_i$ is a number of cells of a given neighborhood, and a number of such rules can be expressed as 2^{L_q}. For CA with e.g. $r = 2$ the length of a rule is equal to $L_q = 32$, and a number of such rules is 2^{32} and grows very fast with L_q. For this reason some other types of rules are used to make them shorter and decrease their total number.

Such a possibility gives e.g. defining a transition function f_t, which updates cells' states on the base of the sum of cells' states in a neighborhood, i.e.

$$q_i^{t+1} = f_t(q_{i-r}^t + ... + q_{i-1}^t + q_i^t + q_{i+1}^t + ... + q_{i+r}^t). \quad (2)$$

For CA with $r = 2$ the length of a rule (called a totalistic rule) is reduced now to $L_t = 6$, and the total number of rules is equal to 2^6.

4 Cellular Automata-based Scheduler

4.1 A concept of CA-based scheduler

To design a scheduling algorithm with use of CA it is assumed that with each task of a program graph an elementary automaton (cell) is associated. A topology of a program graph defines a structure of CA. In opposite to a structure of CA considered in the previous section, the structure of the proposed CA is not regular, however, we still assume that CA is a binary automaton. It results in considering the scheduling problem only for the 2-processor topology: the state 0 or 1 of a cell means that a corresponding task is allocated either in the processor *P0* or *P1*, respectively.

CA corresponding to a given program graph evolves in time according to some predefined rule. Initial states of CA correspond to an initial allocation of tasks in the 2-processor system. Changing states of evolving in time CA corresponds to migration of tasks in the system graph. Changing an allocation of the program graph in the system graph results (under given scheduling policy) in changing the response time T. The question which appears is whether exists a local rule for CA providing for any initial allocation of tasks in the system graph converging CA to an allocation which minimizes T. To solve this problem we need (a) to design properly a local neighborhood (if such a neighborhood is enough to solve the problem) of a cell in the context of the scheduling problem, and (b) find among all possible rules a rule (if such a rule exists) providing a convergence of CA to a solution of the scheduling problem.

Figure 1 presents a concept of CA-based scheduler. There are two phases of operating the scheduler: a phase of learning rules and a phase of normal operating. The purpose of the learning phase is discovery effective rules for scheduling. Searching effective rules is conducted with use of genetic algorithm (GA) [4]. For this purpose an initial random population of rules is created. For a given

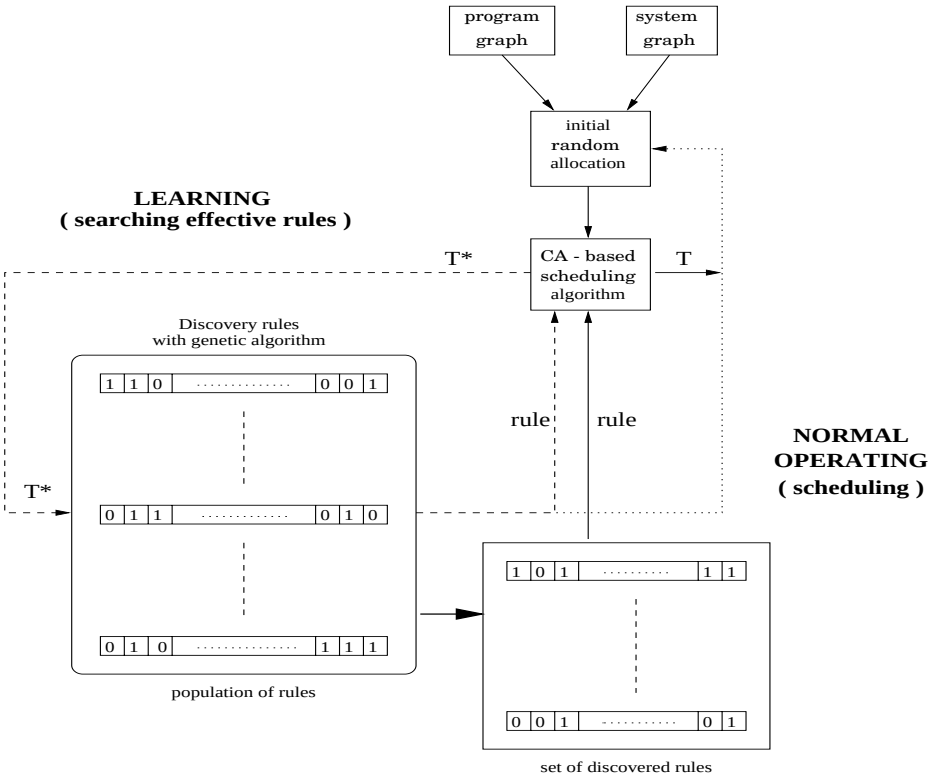

Fig. 1. A concept of cellular automata-based scheduler

random allocation of a program graph into a system graph CA is initialized, and equipped with a rule from the population of rules. CA starts to evolve its states during predefined number of time steps, what results in changing an allocation of task of a program graph. The response time T for a final allocation is evaluated. For a given rule this procedure of evaluation of the rule is repeated a predefined number of times, and it results in evaluation a fitness value T^* for the rule, which is the sum of found values of T corresponding to each repetition of run of CA, and modified to convert a problem of minimization (scheduling problem) to a problem of maximization requested by GA. After evaluation in a similar way all rules from the population, genetic operators of *selection*, *crossover* and *mutation* are involved. Evolutionary process is continued a predefined number of generations, and when is completed discovered rules are stored.

In the phase of normal operating, when a program graph is initially randomly allocated, CA is initiated and equipped with a rule taken from the set of discovered rules. We expect in this phase, that for any initial allocation of tasks of a given program graph, CA will be able to find in a finite number of time steps, allocation of tasks, providing the minimal value of T.

4.2 Designing rules for CA performing scheduling

A neighborhood for an elementary automaton associated with a task of a program graph is created by sets of its predecessors, brothers and successors. Even with such a assumption a size of a potential neighborhood may be large enough and vary depending on a given program graph.

In the proposed neighborhood called a selected neighborhood it is assumed that only two selected representatives of each set of predecessors, brothers and successors will create a neighborhood of a cell associated with a task k. A pair of such representatives of each set is selected on the basis of respectively maximal and minimal values of some attributes of tasks in a given set. We consider a level, a co-level and a computational time b_l as attributes of tasks of all three sets (either $l \in predecessors(k)$ or $l \in brothers(k)$, or $l \in successors(k)$). Additionally, a communication time a_{lk} is also an attribute for tasks which are elements of a set of predecessors and a_{kl} is an attribute of tasks from a set of successors. In a given run of the scheduling algorithm only one attribute for each set is selected. Neighborhood of a cell k associated with a task k defined this way consists of 7 cells, including the cell k.

A neighborhood of a cell k can be defined e.g. in the following way: cells l_1 and l_2 representing tasks from a set of predecessors for which a co-level is maximal or minimal, respectively; cells l_3 and l_4 representing task-brothers for which a computational time (the selected attribute) is largest or smallest, respectively, and cells l_5 and l_6 representing task-successors for which communication times (the selected attribute) from a task k is maximal or minimal, respectively.

Because a structure of a program graph and corresponding CA is irregular, a number of predecessors, brothers or successors may be less than two or they may have the same values of attributes, the following solutions for special cases have been accepted:

- if predecessors (brothers or successors) do not exist for a given task, a neighborhood corresponding to such a situation is still created by a pair of cells; values of these cells (pointing processors where tasks are allocated) are undefined and a state of such a neighborhood will take a special value
- if exists only one predecessor (brother or successor) for a given task, a neighborhood corresponding to this situation is created by a pair of cells; values of these cells will be the same and defined by really existing task-neighbor (a dummy task is allocated on the same processor as a real one)
- if a number of predecessors (brothers or successors) is greater than two and all of them have the same value of an attribute, then cells correspond to two different tasks with the smallest and largest respectively order number assigned to tasks.

After construction of a neighborhood it is necessary to define a state corresponding to each part of this neighborhood. A central cell takes the value 0 or 1. Values of each pair of cells are mapped into one of five values describing a state of the pair in the following way:

- state 0: values of both cells of the pair are the same and equal to 0 (both tasks corresponding to cells are in the processor *P0*)
- state 1: the first cell has the value 0, and the second one has the value 1 (corresponding tasks are in *P0* and *P1* respectively)
- state 2: the first cell has the value 1, and the second one has the value 0 (corresponding tasks are in *P1* and *P0* respectively)
- state 3: values of both cells of the pair are the same and equal to 1 (both tasks corresponding to cells are in the processor *P1*)
- state 4: values of cells are undefined (there is no tasks corresponding to these cells).

A total number of states of a neighborhood can be calculated as 2*5*5*5 and is equal to 250. A length of a rule is 250 bits. A space of solutions of the problem is defined by a number 2^{250} of possible transition functions. GA with a population of rules is used to discover an appropriate rule for CA to solve a scheduling problem.

5 Experiments

In experiments reported in this section it is assumed that CA work sequentially, i.e. at a given moment of time only one cell updates its state. A single run (step) of CA is completed in N_p (N_p - a number of tasks of a program graph) moments of time. A run of CA consists of a predefined number G of steps. A program graph used in the experiment reported in the paper represents the parallel Gaussian elimination algorithm [5] consisting of 18 tasks. All weights of the program graph were scaled dividing them by 10. We refer to this program graph as $gauss18$.

In a learning phase of the scheduling algorithm a population of rules of a size 100 was used, and the learning process was observed during 100 generations. Figure 2 (upper) shows a fitness function $fitt = T^*$ of GA, changing during evolutionary process. Figure 2 (lower) shows a response time T corresponding to the maximal value of the $fitt$, changing during generations.

In the conducted experiments, for each generations of GA a set of four test-problems was created. Each test problem is a random task allocation in the system graph. Each rule from a given population is evaluated on a test problem from point of view of a response time corresponding to a final allocation. For this purpose, CA with a given rule and an initial state corresponding to a given initial allocation is allowed to run a some predefined number of steps. Changing states of CA results in a migration of tasks in the system graph and changing their allocation. To calculate T for a given final allocation of tasks a scheduling policy of the type: a task with the highest value of a dynamic level-first, was applied.

After evaluation of all rules from a given population, GA operators are applied. A proportional selection with elitist strategy was used. A crossover with a probability $p_c = 0.95$, and a bit-flip mutations with $p_m = 0.001$ were used.

Figure 3a shows a run of CA-based scheduler with the best rule found in the 5-th generation. Left part of the figure presents a space-time diagram of

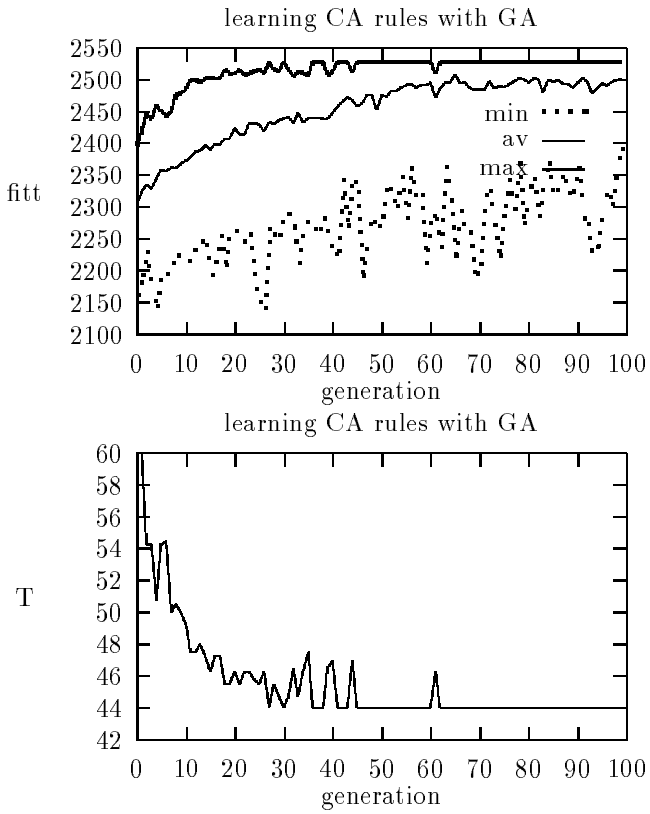

Fig. 2. Discovery by GA rules for CA-based scheduler operating with *gauss*18: evolutionary processing population of rules (upper), and response time T improved during the evolutionary process (lower)

CA consisting of 18 cells, and the right part shows graphically a value of T corresponding to an allocation found in a given step. One can see that after the step 0, cells of CA are in some states corresponding to allocation of tasks (white cell - a corresponding task is allocated in $P0$, black cell - a task is allocated in $P1$), and the value of T corresponding to this allocation is greater than 81. After few steps, CA starts to oscillate, repeating a sequence of six states with resulting patterns of task allocation, and corresponding changing values of T. In generation 46, GA discovers (see Figure 2(lower)) a rule providing an allocation with an optimal value $T = 44$. Such a value was found in [5], but it needed three processors, while CA finds a solution with two processors.

The found rule is, however, not absolutely the best. The rule does not pass a test on a test problem created in generation 62 (see, Figure 2(lower)). GA quickly modifies this rule and it passes successfully all subsequent tests. Figure 3b shows a space-time diagram of such a rule existing in the generation 100.

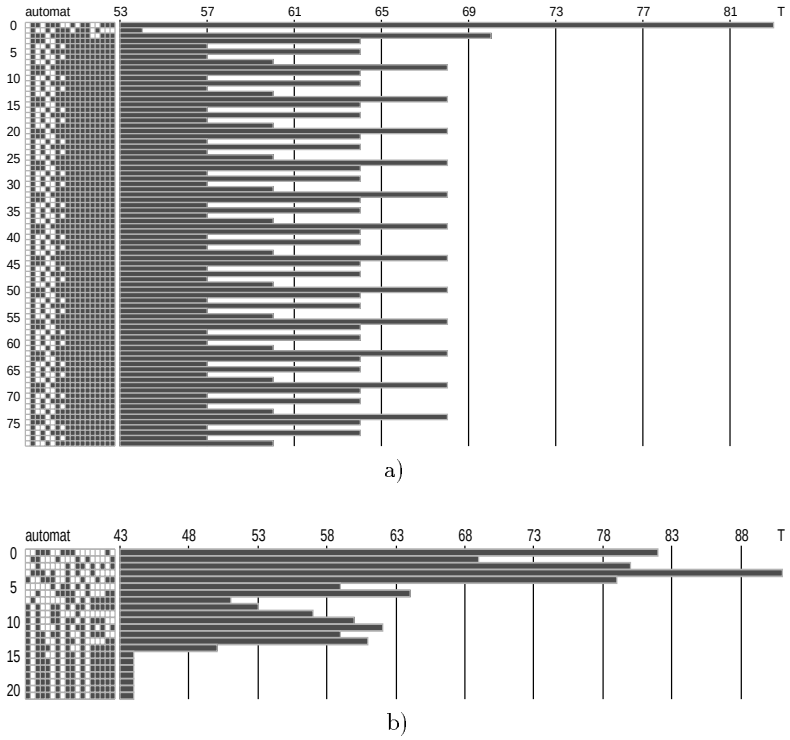

Fig. 3. Space-time diagrams of CA-based scheduler with the best rule found for *gauss*18 in generation 5 (a), and generation 100 (b)

After run of GA its population contains rules suitable for CA-based scheduling. How good are these rules? We can find out it in the process of normal operating of CA. We generate some number of test problems, and use them to test each of found 100 rules. Figure 4 shows results of the test conducted with 100 random initial allocation of the *gauss*18. For each rule from the population (before the test rules were sorted from the greatest to the smallest values of their fitness function) the average value of T (avr T) found by CA in the same test problems is shown. One can see that 29 found rules provide the maximal efficiency of the scheduling algorithm.

6 Conclusions

Results concerning ongoing research on development of CA-based distributed algorithms of scheduling tasks of parallel programs in parallel computers have been presented in the paper. The results of conducted experiments show that GA is able to discover for a given instance of a problem effective rules for CA-based

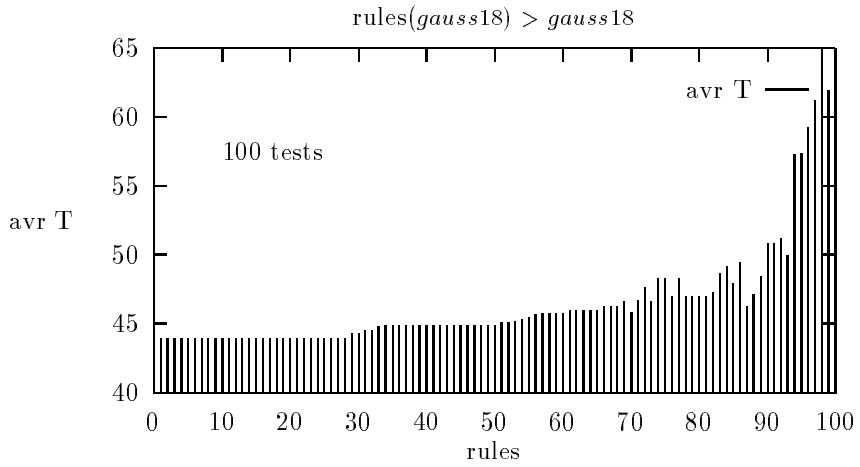

Fig. 4. Testing efficiency of discovered CA rules for the *gauss*18

scheduler. It is worth noticing that the scheduling conducted by CA during normal operating is *fully distributed*. Decisions concerning changing states of cells are taken observing only local neighborhood of a given task. In subsequent steps of operating CA none estimation of T is calculated. This value is calculated only by an external observer of the scheduling process conducted by CA.

References

1. I. Ahmad (Ed.). Special Issue on Resource Management in Parallel and Distributed Systems with Dynamic Scheduling: Dynamic Scheduling, *Concurrency: Practice and Experience*, 7(7), 1995
2. R. Das, M. Mitchell, and J. P. Crutchfield, A genetic algorithm discovers particle-based computation in cellular automata, In Davidor, Y., Schwefel, H.-P., Männer, R. (Eds.). *Parallel Problem Solving from Nature – PPSN III*. LNCS 866, Springer, 1994.
3. H. El-Rewini, T. G. Lewis, and H. H. Ali, *Task Scheduling in Parallel and Distributed Systems*. PTR Prentice Hall, 1994.
4. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989
5. 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.
6. S. Wolfram, Universality and Complexity in Cellular Automata, *Physica D* 10, 1-35, 1984

This article was processed using the LaTeX macro package with LLNCS style