SCAS-IS: Knowledge Extraction and Reuse in Multiprocessor Task Scheduling based on Cellular Automata

Murillo G. Carneiro
Institute of Mathematical Sciences and Computing, ICMC
University of São Paulo, USP
São Paulo, Brazil
carneiro@icmc.usp.br

Gina M. B. Oliveira
Faculty of Computing, FACOM
Federal University of Uberlândia, UFU
Uberlândia, Brazil
gina@facom.ufu.br

***Abstract*—Static Task Scheduling Problem (STSP) in multiprocessors is a NP-Complete problem. Cellular Automata (CA) have been recently proposed to solve STSP. The main feature of CA-based models to STSP is the extraction of knowledge while scheduling an application and its subsequent reuse in other instances. Previous works showed this approach is promising. However some desirable features have not been successfully exploited yet, such as: (i) the usage of an arbitrary number of processors, (ii) the massive parallelism inherent to CA and (iii) the reuse of evolved rules with competitive results. This paper presents a new model called SCAS-IS (Synchronous Cellular Automata Scheduler with Initialization Strategies). Its major innovation is the employment of fixed initialization strategies to start up CA dynamics. Parallel program graphs found in literature and others randomly generated were used to test the new model. Results show SCAS-IS overcame related models both in makespan obtained as computational performance. It is also competitive with meta-heuristics.**

***Keywords*-cellular automata; multiprocessor task scheduling; genetic algorithms.**

I. Introduction

The increasing availability of parallel and distributed systems turns possible the development of computationally-complex and time-consuming applications. However, to better explore the potential of such computational power it is necessary to develop efficient methods and tools for managing this resource. Task scheduling plays a key role in multiprocessor architectures. The split of an application into independent tasks and their allocation among processors are critical for an efficient employment of such environments.

In a broad sense, scheduling is a decision-making process that involves resources and tasks in the search for optimize some objective, typically the resultant runtime or makespan [1]. Multiprocessor task scheduling is the process to allocate a set of independent computational tasks in a parallel application into processing nodes of the architecture. In the present work, a version of this problem is considered in which all information about the tasks is known a priori. It is also known as static task scheduling problem (STSP) [2]. Given an instance of STSP, represented by a DAG (directed acyclic graph), if precedence constraints among tasks are satisfied and the resultant makespan is minimized we have an optimal scheduling. Even limited to the simplest case - a parallel system with only two processors - the problem to find an optimal scheduling is NP-Complete [3]. There are several works in the literature investigating approaches to solve STSP which typically employ heuristics or meta-heuristics [2], [4]. Genetic algorithms and simulated annealing are often explored to this problem [4].

Cellular Automata (CA) are simple discrete dynamical systems composed by lattice and transition rule. The employment of CA models to solve STSP was previously investigated in [5]–[9]. In such works, an evolutionary method is employed in a learning phase to found adequate rules to schedule a parallel program. The possibility of implementation on parallel hardware together with the simplicity of its basic components are among the most notable features of cellular automata [10]. These characteristics turn them adequate to be implemented in massive parallel architectures like FPGA speeding up the throughput. However, almost all previous models presented in literature were not able to explore the inherent parallelism into CA because they use an asynchronous (sequential) updating of cell states (only one cell can update its state at a time) [5]. Recently, a scheduler model called Synchronous Cellular Automata Scheduler (SCAS) [9] has presented good scheduling results while employing a synchronous updating of cells, turning its parallel implementation viable.

Using traditional heuristics or meta-heuristics approaches to STSP, a computational effort is used to solve an instance of the problem and when a new instance is presented to the algorithm, the process need to start again from scratch. A promising skill of CA-based previous approaches is their ability to extract knowledge from the scheduling process of a parallel application and reuse it in others instances. However it was identified limitations in previous studies specially in the reuse phase and when a number of processors above two is used in the system architecture. In our investigations we discover that one reason of such weakness is due to the guideline that CA transition rules evolved in the learning phase must be able to perform the schedule starting from any initial configuration of the lattice. This lack of sensibility required in the previous models turns the evolutionary search complex and time-consuming and it does not return any actual contribution to scheduling process.

A new model called Synchronous Cellular Automata-based Scheduler using fixed Initialization Strategies (SCAS-IS) is presented and evaluated here. The major innovation of the new model is the employment of a few number of simple initialization strategies to start up CA dynamics instead of the usage of arbitrary random initial configurations (ICs). As a consequence the evolved rules need to be able to perform the schedule starting only from a small number of ICs given by deterministic initialization strategies. In this work, to establish the initial allocation of tasks, three simple initializations are used. As its predecessor SCAS [9], the resultant method uses a linear neighborhood and it is suitable to be implemented in parallel hardware since it employs a synchronous cells updating. Results show that SCAS-IS model was able to perform efficient task scheduling in systems with more than two processors and it was able to extract knowledge during the process of scheduling of an application and reuse it to solve other instances. It also overcame previous CA-based models.

The remainder of this paper is organized as follows: Section II presents a backgroundabout multiprocessor scheduling, cellular automata and CA-based scheduling models. Section III presents some analysis about the previous models and describes the proposed model: SCAS-IS. Section IV contains experimental results concerning SCAS-IS and comparing it with other methods. The last section contains conclusions about the current research and some propositions of future investigations.

II. Background

A. Static Task Scheduling Problem

A parallel program can be represented by a directed acyclic graph (DAG) defined by tuple $G_P = (V, E, W, C)$, where $V = \{t_1, \ldots, t_N\}$ denotes the set of N graph tasks (nodes); $E = \{e_{i,j} \mid t_i, t_j \in V\}$ represents the set of communication edges, also called precedence constraints; $W = \{w_1, \ldots, w_n\}$ represents the set of tasks runtime, in others words, for each task $t \in V$ is assigned a computational weight $w(t) \in W$ relative to its computational cost; and $C = \{c_{i,j} \mid e_{i,j} \in E\}$ denotes the set of communication times of the edges, in others words, for each edge $e_{i,j} \in E$ is assigned a communication cost $c_{i,j} \in C$ related to the cost of data transfer between tasks t_i and t_j when running on different processors. G_P is called program graph. Figure 1 shows an example of program graph called *gauss18* that represents a set of 18 tasks.

In multiprocessor model considered here, all processors have the same computational power and the communications between the channels do not consume any time of the processor. Furthermore, a scheduling policy defines

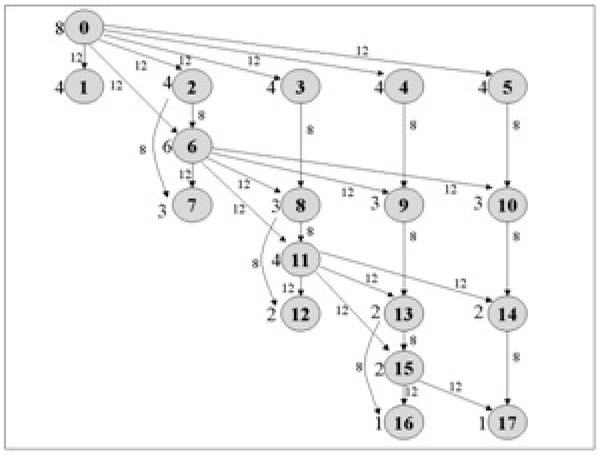

Figure 1. Example of program graph (*gauss18*).

the running order of tasks in each processor. Note that while the scheduler distributes tasks among processors, the scheduling policy ordering these tasks within each processor. The scheduling policy used in this work is: the task with the highest dynamic b-level first [5]. The b-level is the higher cost between a task and an exit node. The b-level is dynamic when is calculated considering the allocation of the tasks in processors and the communication cost is only considered when tasks are distributed on different processors.

B. Cellular Automata

CA is a discrete dynamic system composed of a cellular space and a state transition function [11]. The cellular space is a lattice of l cells (simple and similar components that have local connectivity and boundary conditions) organized in an d-dimensional arrangement. Each cell assumes a state from a finite set of k possible states in each time step. The transition function or rule f determines the next state of each cell. Temporal evolution is the process of applying the rules over the lattice by a given number of time steps T. Cells updating usually happens in the following ways:

- **Parallel or Synchronous**: in which all cells of the lattice update their states synchronously at each time step.
- **Sequential or Asynchronous**: in which only one cell updates its state at each time step and this new state is considered in updating of others cells.

A neighborhood of radius R is defined for each cell in the lattice. The neighborhood length of a cell i is given by $m = 2R+1$ and the next state of a cell is given by a function of its current state and the current states of their neighbors in the lattice (R cells to each side). Figure 2(a) shows a one-dimensional ($d = 1$) and binary ($k = 2$) cellular automaton with 5 cells ($l = 5$) and neighborhood of radius 1 ($m = 3$). Figure 2(b) shows a transition rule (f) that presents the new state of central cell for all possible configurations of the neighborhood. Figure 2(c) presents the temporal evolution of the lattice by two time steps using synchronous updating and periodic boundary condition (the lattice is a ring). Figure 2(d) shows the temporal evolution of lattice by two time steps using sequential updating and null boundary condition (both the neighbor to the left of first cell as the neighbor to the right of the last cell are considered in the state 0).

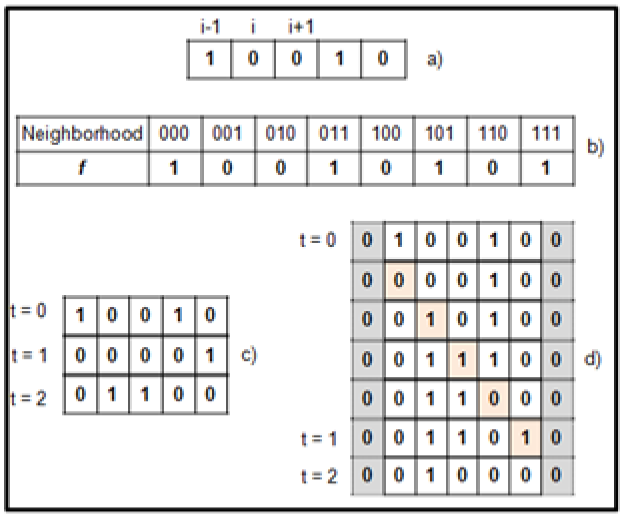

Figure 2. CA example: (a) initial lattice; (b) transition rule; (c) parallel evolution mode; (d) sequential evolution mode.

C. Cellular Automata-based Scheduler

A CA-based scheduler model was presented in [5] in which it is assumed that each cell of the lattice is associated with a task. If a set of tasks has cardinality x, CA lattice has x cells. Furthermore, given an architecture consisting of w processors, CA will have w possible states. Assuming a system with two processors ($P0$ and $P1$), each cell can take value 0, indicating that the corresponding task is allocated on processor $P0$, or value 1 ($P1$). The model operates in two modes: learning and operation. It is a hybrid model because it employees a genetic algorithm (GA) to learning CA rules.

In the learning mode, GA is used to search for rules able to converge the lattice to optimal (or sub-optimal) allocations of a given DAG, starting from random initial configurations. GA population (P) is initially formed by CA state transition rules randomly generated. Fitness function is calculated in each GA generation by: (i) randomly sorting a set of initial lattices S_{Latt} representing random allocations of tasks in processors; (ii) temporal evolution of each lattice l of S_{Latt} by each rule transition r in P for S time steps; (iii) allocations obtained in time S are ordered in each processor using a scheduling policy obtaining makespan associated to each pair (r, l); (iv) rule fitness is given by the average of makespan calculated starting from each lattice of S_{Latt}. The best rule presents the smallest makespan average.

In the operation mode it is expected that, for any initial allocation of tasks, CA rules stored after learning are able to evolve the lattice until a configuration which represents an optimal allocation, that is, it minimizes the makespan. It is also desired that the rules obtained in the learning phase can be used in the scheduling of other graphs.

III. SCAS-IS

A. Analysis of related models

The majority of works related to CA-based scheduling used only two processors [5], [7]–[9]. An attempt to increase the number of processors (V_s) was presented in [6]. However, it showed that the scheduler model is not able to deal with the complexity increasing due to the usage of number of processor greater than two. Table I shows the results published in [6] using CAS model with sequential updating in the learning phase which are compared with the results obtained by a genetic algorithm (GA). The second model is a "pure" genetic algorithm that allocates and schedules the tasks in processors. That is, such model does not use CA rules. Table I also shows results of a similar experiment we performed using SCAS environment (presented in [9]) in the learning phase. As one can see the results of CAS and SCAS CA-based models starting from $V_s = 3$ decays when compared to meta-heuristic GA. The worst results were obtained using program graph *gauss18*.

Table I
BEST MAKESPAN FOUND USING GA, CAS AND SCAS MODELS FOR DIFFERENT PROGRAM GRAPHS.

G_P	CAS [6]			SCAS			GA [6]		
	2	3	4	2	3	4	2	3	4
g18	46,0	38,0	27,0	46,0	38,0	27,0	46	36	26
g40	80,0	57,0	46,0	80,0	57,0	45,2	80	57	45
gauss18	44,0	48,0	52,0	44,0	52,0	50,8	44	44	44

We performed new comparative experiments to evaluate previous models CAS and SCAS in the operation phase, when the rules learned for one specific graph (*gauss18*) are applied to schedule other programs graphs. We implemented CAS and SCAS with linear neighborhood being that the first uses sequential updating as in [6] and the second employs parallel updating as in [9]. Table II shows results of rules learned based on *gauss18* using CAS and SCAS when applied to graphs *g18*, *g40* [6] and three random graphs with 30, 40 and 50 tasks [9]. We also implemented a pure GA and a simulated annealing (SA) for scheduling to use their results as reference values. One can also observe that results of reusing *gauss18* rules are not reasonable when compared with meta-heuristics for both CA-based models, being that SCAS is even worst than CAS. In [9], the models were investigated only in learning phase and CAS and SCAS were similar. We believe that a possible cause to the undesirable performance of previous models in reuse phase are related to the way in which CA lattices are initialized to start scheduling as explained in next section.

Table II
REUSE OF THE RULES EXTRACTED FOR *gauss18* ON DISTINCT PROGRAM GRAPHS IN SCAS, CAS AND SCAS-IS MODELS FOR $V_s = 2$.

G_P	**GA**	**SA**	**CAS**	**SCAS**	**SCAS-IS**
g18	46	46	49,49	64,05	46
g40	80	82	86,25	106,40	80
rand30	1222	1222	1336,61	1743,25	1230
rand40	983	997	1136,55	1268,29	990
rand50	624	664	730,70	837,22	650

B. Description of SCAS-IS

In previous works [5]–[9], the scheduler model focuses on the capacity of a transition rule to evolve any random initial lattice to a configuration that represents the optimal allocation of tasks. Thus, during the learning phase, each rule is evaluated according to its performance when scheduling a set of initial lattices (S_{Latt}). Besides, in the normal phase, the quality of any evolved rule is measured using it to schedule a new set of random lattices. However, the search for this independence to initial lattice makes the rules search complex and computationally intensive, embarrassing GA convergence. Furthermore, we believe that the more important generalization is not related to the randomly initial lattice used to start the scheduling process, instead it is related to the capacity of a rule to schedule others instances in reuse mode. We build a new model named Synchronous Cellular Automata-based Scheduler using fixed Initialization Strategies (SCAS-IS) to deal with these complexities. The major innovation of the new model proposed here is the employment of a few number of simple initialization strategies to start up CA evolution instead of the usage of arbitrary random initial configurations. As a consequence the evolved rules need to be able to perform the schedule starting only from a small number of initial configuration of the lattices, given by a deterministic and simple initialization strategy. In this work, to establish the initial allocation of tasks, three simple initializations are used: IS_1, IS_2 e IS_4. IS_1 establishes a state switching of cells, in a lexicographic order of states, one-by-one, starting from the first lattice cell. If the lattice size is not a multiple of 2, the last cell is truncate in state 0. For example, for *gauss18* using $V_s = 2$, as the lattice has 18 cells and 2 possible states (0 and 1) the initial lattice given by IS_1 is: 010101010101010101. IS_2 and IS_4 are similar strategies being that the first switches states two-by-two and the second four-by-four. For *gauss18* with $V_s = 2$, IS_2 and IS_4 give the initial lattices 001100110011001100 and 000011110000111100, respectively.

SCAS-IS employees synchronous updating of cells (as SCAS) and works in two modes: learning and reuse. It receive as input V_s and a program graph. Initialization strategies are used to create three initial allocations. The learning mode is based on SCAS: it employs tournament selection and fitness-based reinsertion, the final population (parents and children) is ordered and the best rules are selected to next generation. As pointed in [9], these strategies stimulate the competition between individuals allowing a widely search in solution space, which is very difficult using elitist strategy and parallel updating as in [5]–[8]. The main steps in evaluation are: (i) applying fixed initialization strategies chosen a priori to obtain three initial allocation which defines the IC of the three lattices; (ii) temporal evolution of the lattices using each rule transition r in P for S time steps; (iii) the final lattices in time S define the final allocations of tasks which are ordered in each processor using a scheduling policy obtaining makespan associated to each rule r; (iv) rule fitness is equal to smaller makespan between three final allocations obtained using it. The best rule in P presents the smallest makespan. In fact, step (i) described above is performed only once during GA run, because the allocation determined by initialization strategies are fixed and they are used in all evaluations. In reuse mode, the CA is equipped with a set of rules in repository RDB and SCAS-IS receives a new program graph to schedule. The same ISs used in learning mode are used in reuse mode in step (i) to obtain the initial allocation associated to the graph. Steps (ii) to (iv) are executed for each rule in RDB, returning the scheduling with the smallest makespan.

IV. RESULTS AND DISCUSSION

Experiments to evaluate SCAS-IS performance are discussed here. The results obtained are exhibited in the first column of Table III. They are compared with reproductions of other CA-based approaches: CAS [6] and SCAS [9]. A drawback related to these previous CA-based schedulers is that they not ensure the same makespan in learning and operation mode for a given program graph since lattice ICs are randomly generated and the rules may not evolve them to the same results found by GA evaluation as one can see in Table III. On the other hand, SCAS-IS does not need an operation mode because it always starts from the same ICs obtained by ISs. In addition, the table also presents the best results obtained using ISs, which can be thought as the kick-off for SCAS-IS. The table also presents results of two meta-heuristics applied to this problem: genetic algorithm (GA) and simulated annealing (SA). Columns "BEST" represents the smallest makespan found (out of 20 runs) for GA, SA and CA-based models (which represents the smallest makespan found when testing all rules found in learning mode) and columns AVG shows the makespan average considering all runs (learning phase to CA-based approaches). Considering CAS and SCAS results, CA rules was applied starting from 1000 random ICs of the lattice, as in normal operation mode [6], and the table (column BEST) shows the smallest average obtained by the best rule. Considering SCAS-IS results, CA rules were applied starting

Table III
MAKESPAN RESULTS FOUND BY THE BEST CA RULES EVOLVED WITH SCAS-IS, CAS AND SCAS (LEARNING MODE) COMPARED WITH THOSE FOUND BY META-HEURISTICS GA AND SA.

G_P	V_s	**SCAS-IS**		**CAS**		**SCAS**		**GA**		**SA**		**IS**	**T**
		BEST	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST	
g18	2	46	46,00	46,00	46,00	46,00	46,00	46	46,00	46	46,00	46	=
	3	36	36,35	38,00	38,36	38,00	38,47	36	36,00	36	36,10	40	$<$
	4	26	26,05	26,00	26,90	27,00	26,99	26	26,00	26	26,00	27	$<$
g40	2	80	80,00	80,00	80,76	80,00	80,81	80	80,00	82	86,90	81	$<$
	3	57	57,00	57,00	57,15	57,00	57,22	57	57,15	66	67,85	58	$<$
	4	45	45,45	45,28	45,82	45,20	45,92	46	46,45	55	57,75	46	$<$
gauss18	2	44	44,55	44,00	47,60	44,00	47,86	44	44,95	44	46,30	70	$<$
	3	44	45,90	52,00	52,65	52,00	52,21	44	45,65	44	46,15	75	$<$
	4	44	45,55	52,00	52,68	50,80	52,44	44	45,65	46	47,35	75	$<$
rand30	2	1222	1224,00	1239,00	1247,31	1226,95	1267,50	1222	1222,05	1222	1244,60	1280	$<$
	3	894	934,75	963,00	1021,89	1012,86	1024,39	821	860,90	970	1027,05	1025	$<$
	4	822	839,65	911,19	1011,58	986,00	1020,39	753	785,35	853	914,70	1023	$<$
rand40	2	983	985,10	1006,00	1020,47	997,27	1024,19	983	983,15	997	1046,65	1056	$<$
	3	710	724,50	810,94	833,40	796,60	849,27	685	699,05	794	861,35	907	$<$
	4	608	624,95	681,00	719,65	725,24	762,76	561	585,55	684	759,90	743	$<$
rand50	2	624	638,60	661,18	667,78	662,90	673,98	624	626,80	664	709,00	684	$<$
	3	580	605,20	643,09	659,60	655,12	668,88	504	532,60	624	680,20	784	$<$
	4	580	603,60	620,00	643,98	642,64	661,51	508	528,80	600	671,80	724	$<$

from three IC lattices (IS_1, IS_2 and IS_4) and the table shows the makespan obtained by the best rule.

Results in Table III show a good advantage of SCAS-IS over other CA-based approaches. Column "T" shows the results of Mann-Whitney test between SCAS-IS and CAS considering all runs. There is statistical evidences that SCAS-IS is better than CAS in 17/18 cases considering a significance level of 95%. A similar test between SCAS and SCAS-IS was performed and there is also statistical evidences that SCAS-IS overcame its predecessor SCAS in the same 17 cases. An analysis of SCAS-IS performance in relation to meta-heuristics GA and SA can also be performed using Table III. Considering graphs extracted from literature, SA returned the worst results while GA and SCAS-IS returned best results. Considering random program graphs, GA overcame SCAS-IS, especially for $V_s = 4$, but the new model was able to improve the value obtained by initialization strategies (ISs) and it was better than SA.

SCAS-IS was also evaluated in reuse. CA was equipped with best rules extracted in learning mode for *gauss18*. Such rules were applied to schedule other program graphs. The last column of Table II shows the best scheduling found by SCAS-IS rules in reuse mode. The best values were obtained by GA but it must be take into account that this model was evolved for each one of the graphs, while SCAS-IS was evolved only for *gauss18*. Finally, comparing SCAS-IS with CAS and SCAS, it is possible to observe there is a significant improvement in new model in relation to the previous ones.

We performed a second type of experiment related to reuse mode using the best rules found by CA-based models in the learning mode with *gauss18* as the target graph. They were applied in 10 distinct variations of this program graph. These variations were presented in [7]; they are program graphs with 18 tasks very similar to *gauss18*. In [7], CA-based models with non-linear neighborhoods were evaluated. Such kind of neighborhoods are much complex than the linear used in CAS, SCAS and SCAS-IS, and they have presented a good performance to deal with non-linear graphs as gauss18. The best approach evaluated in [7], named here "Joint-CAS" uses a joint evolution in learning phase, where 5 program graphs were used together with *gauss18* to search for more generalized rules. Table IV presents a comparative scheme between SCAS-IS, CAS and Joint-CAS methods, besides meta-heuristics GA and SA. In learning mode, SCAS-IS, Joint-CAS and CAS search for rules to program graph *gauss18*. GA and SA presented the best averages, but the advantage of SCAS-IS over meta-heuristic algorithms is that the evolutionary process occur only once and the knowledge extracted is used to schedule others program graphs. Thus, the computational time is reduced (by 100 times approximately). SCAS-IS overcame CAS in all instances, returning a better average. Performance of SCAS-IS is still better than Joint-CAS in average. Besides, one must take into account that Joint-CAS evolves 6 graphs in learning consuming extra time and it uses sequential updating and a complex and costly nonlinear neighborhood.

V. CONCLUSIONS

Our investigation about published CA-based scheduling models have pointed some drawbacks in previous works: (i) an inefficient employment of synchronous updating of CA cells driving previous studies towards the usage of sequential updating embarrassing fast parallel implementation of such models; (ii) a difficult to employ simple linear models of neighborhood what was a barrier to use an arbitrary number

Table IV
COMPARATIVE ANALYSIS BETWEEN RESULTS OBTAINED BY META-HEURISTICS AND CA-BASED MODELS IN REUSE MODE.

G_P	**AG**	**SA**	**CAS**	**Joint-CAS**	**SCAS-IS**
gauss18-6	47	47	48	47	48
gauss18-7	44	44	44	47	44
gauss18-8	44	46	47	47	47
gauss18-9	46	46	48	47	47
gauss18-10	44	45	44	47	44
gauss18-11	46	46	51	47	47
gauss18-12	47	47	48	47	48
gauss18-13	44	45	47	47	45
gauss18-14	46	46	48	47	47
gauss18-15	46	46	48	47	47
Average	45,40	45,80	47,30	47,00	46,40

of processors (iii) non effective reuse of evolved rules which is the crucial point to justify the employment of cellular-automata models in scheduling. In the present work a new CA-based scheduler model named SCAS-IS (Synchronous Cellular Automata-based Scheduler using fixed Initialization Strategies) is presented. Some advantages of the new model: (i) employment of synchronous updating with results competitive with previous models based on sequential updating (ii) usage of linear neighborhood which able the model to use more than 2 processors in the architecture with good results in the learning phase (iii) usage of a new and simple approach to initialize CA lattices which turns the model more likely to extract knowledge during a learning stage with a posteriori use of this knowledge to solve new instances. The major innovation is the usage of three simple initialization strategies to start CA evolution instead of the employment of a sample of random initial lattices as performed in previous models (typically 1000 random lattices were used previously). Apparently, the independence to initial lattices turned previous models unable to be reused in new instances of program graphs after the learning phase using a target graph, since the CA rules evolved in learning phase tend to evolve lattice to the same final configuration: the one which optimizes the target graph. Experiments confirmed SCAS-IS has much better performance to reuse CA rules compared to previous related models. The scalability of the new model in respect to the number of processors was investigated. Results show that SCAS-IS overcame related models in learning phase when the number of processor is increased above two. In addition, SCAS-IS is competitive with the best results found by meta-heuristics. The computation cost of the new model was also reduced when compared to others models. Forthcoming works include the analysis of SCAS-IS faced to a high number of nodes in multiprocessor architecture and the investigation of new models of neighborhoods.

ACKNOWLEDGMENT

M.G.C. thanks to CNPq and FAPESP for financial support and G.M.B.O is grateful to CNPq and FAPEMIG.

REFERENCES

[1] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 3rd ed. Springer Science, 2008.

[2] Y. K. Kwok and I. Ahmad, "Benchmarking and comparison of the task graph scheduling algorithms," *Journal of Parallel and Distributed Computing*, vol. 59, no. 3, pp. 381–422, 1999.

[3] M. R. Garey and D. S. Johnson, *Computers and Interactability. A Guide to the Theory of NPCompleteness.* Freemann And Company, 1979.

[4] S. Jin, G. Schiavone, and D. Turgut, "A performance study of multiprocessor task scheduling algorithms," *The Journal of Supercomputing*, vol. 43, pp. 77–97, 2008.

[5] F. Seredynski and A. Y. Zomaya, "Sequential and parallel cellular automata-based scheduling algorithms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 10, pp. 1009–1022, 2002.

[6] A. Swiecicka, F. Seredynski, and A. Y. Zomaya, "Multiprocessor scheduling and rescheduling with use of cellular automata and artificial immune system support," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 3, pp. 253–262, 2006.

[7] P. M. Vidica and G. M. B. Oliveira, "Cellular automata-based scheduling: A new approach to improve generalization ability of evolved rules," *Brazilian Symposium on Artificial Neural Networks (SBRN'06)*, pp. 18–23, 2006.

[8] G. M. B. Oliveira and P. M. Vidica, "A coevolutionary approach to cellular automata-based task scheduling," *Lecture Notes in Computer Science (accept to ACRI Conference - Cellular Automata for Research and Industry)*, 2012.

[9] M. G. Carneiro and G. M. B. Oliveira, "Cellular automata-based model with synchronous updating for task static scheduling," in *Proceedings of 17th International Workshop on Cellular Automata and Discrete Complex System*, 2011, pp. 263–272.

[10] M. Sipper, *Evolution of Parallel Cellular Machines, The Cellular Programming Approach.* Springer, 1997.

[11] P. Sarkar, "A brief history of cellular automata," *ACM Comp. Surveys*, vol. 32, no. 1, pp. 80–107, 2000.