Natural Computing Journal manuscript No.
(will be inserted by the editor)

Synchronous Cellular Automata-Based Scheduler initialized by Heuristic and modeled by a Pseudo-linear neighborhood

Murillo G. Carneiro · Gina M. B. Oliveira

the date of receipt and acceptance should be inserted later

Abstract Cellular Automata (CA) are able to produce a global behavior from local interactions between their units. They have been applied to the task scheduling problem in multiprocessor systems in a very distinguished way. As this problem is NP-Complete, heuristics and meta-heuristics are usually employed. However, these techniques must always start the scheduling process from scratch for each new parallel application given as input. On the other hand, the main advantage to use CA for scheduling is the discovery of rules while solving one application and their subsequent reuse in other instances. Recently studies related to CA-based scheduling have shown relevant approaches as the use of synchronous updating in CA evolution and good results in multiprocessor systems with two processors. However, some aspects, such as the low performance of CA-based schedulers in architectures with more than two processors and during the reuse of the discovered rules, need to be investigated. This paper presents two new models to improve CA-based scheduling to deal with such aspects. The first proposal refers to the employment of a construction heuristic to initialize CA evolution and the second one is a new neighborhood model able to capture the dependence and relations strength among the tasks in a very simple way. It was named pseudo-linear neighborhood. An extensive experimental evaluation was performed using graphs of parallel programs found in the literature and new ones randomly generated. Experimental analysis showed the combined application of both techniques makes the search for CA transition rules during learning stage more robust

M. G. Carneiro
Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo
Avenida Trabalhador São-Carlense, 400, 13566-590, São Carlos, Brasil
E-mail: carneiro@icmc.usp.br

G. M. B. Oliveira
Faculdade de Computaccão, Universidade Federal de Uberlândia
Avenida João Naves de Ávila, 2.121, 38400-902, Uberlândia, Brasil
E-mail: gina@facom.ufu.br

and leads to a significant gain when considering the reuse of them on real-world conditions.

1 Introduction

Cellular automata are dynamical systems in which time, space and variables are discrete, turning them adequate to be applied as models in different computational challenges [4]. Wolfram's works about CA dynamics [18,19,21,22] revealed that even the simplest CA models can represent interesting patterns and exhibit emergent behaviors. Therefore, they have been studied in a growing range of problems and applied to tasks related to pattern recognition [13], criptography [23], scheduling [14,15,2,10,1,3], complex systems and artificial life simulation [20,13,6].

A cellular automaton is composed of a d-dimensional arrangement of simple local units - or cells - and a state transition function also called transition rule. CA are able to produce a global behavior from local interactions between their units and exploit highly parallel architectures such as FPGA (Field-Programmable Gate Arrays) [17]. Based on these features, this paper investigates an interesting CA-based approach related to its application to task scheduling [2], which plays a central role in multiprocessor architectures. A promising skill of such an approach is the CA ability to extract knowledge from the process when scheduling an instance of parallel program and reuse it to schedule other instances. On the other hand, traditional heuristics and meta-heuristic approaches to this problem require high computational effort to solve each new instance of the problem. Therefore, the key motivation to study CA-based scheduling is the possibility to discover transition rules presenting generalization ability, so that they can be used to schedule different instances without the need of a new scheduling process from scratch.

Starting from a previous model [14] successively refined in [15,16,2,1,3], new investigations are discussed here to improve the CA-based scheduling performance. Previous CA-based scheduler models have a learning phase in which a genetic algorithm is applied to search for CA transition rules able to schedule a specific program graph. The major goal is to find CA rules adequate not only to schedule the program graph used as target but also to be applied to other unseen program graphs.

The first investigation here refers to the employment of a construction heuristic to initialize the CA evolution. A preliminary analysis was presented in [3], where results for architectures with two processors were discussed. Scheduling results were highly improved in comparison to those of previous approaches [15]. This model was named SCAS-H: Synchronous Cellular Automata-Based Scheduler initialized by Heuristic. However, later results, specially considering multiprocessor systems with more than two processors, have shown that although the usage of construction heuristics indeed improves makespan in the learning phase of the CA-based scheduler, the improvement was not so emphatic in the reuse phase, when the learned rules are applied to new instances

of program graphs. Subsequent analysis led us to conclude that this limitation to manipulate more than two processors is partially related to the simple linear neighborhood model employed in previous models, whose neighbor relations are defined based only on the order number of tasks. This observation motivated the second investigation reported here: a new model able to capture the spatial relations of the computational tasks in a very simple way. It was named here pseudo-linear neighborhood, since it preserves the simple structure of linear neighborhoods, but the neighbors relations are defined by the proximity and relative importance of the tasks within the program graph. Finally, experiments showed the combined application of both techniques - initialization by construction heuristics and pseudo-linear neighborhood - makes the search for CA transition rules during learning more robust, leading to a significant gain when considering the reuse of them on real-world conditions. We called the resultant scheduler model SCAS-HP: Synchronous Cellular Automata-Based Scheduler initialized by Heuristic and modeled by a pseudo-linear neighborhood.

The remainder of the paper is organized as follows: Sect. 2 defines the general concepts and related works about CA for scheduling; Sect. 3 presents the proposed CA-based scheduler models related to initialization heuristic and pseudo-linear neighborhood; Sect. 4 provides computer simulation results to analyze SCAS-H and SCAS-HP models. Moreover, this section shows the new techniques can really improve the CA-based scheduling on real world conditions; Finally, Sect. 5 concludes the paper.

2 Models based on CA for task scheduling

This section offers a general background of the use of CA-based models for task scheduling. It is organized as follows:

- Cellular automata, which plays the key role in CA-based scheduling is described in Subsect. 2.1.
- Task scheduling, which is a NP-Complete problem and a formulation about its can be seen in Subsect. 2.2.
- The most important elements in CA-based scheduling and the state of the art related to previous approaches are presented in Subsect. 2.3.

2.1 Cellular Automata

Basically, a cellular automaton consists of the cellular space and the transition rule. Cellular space is a regular lattice of η cells, each one with an identical pattern of local connections to other cells, and subjected to some boundary conditions. These cells are arranged in a d-dimensional space and the most studied are the one-dimensional and the two-dimensional arrangements. Each cell assumes a state from a finite set of κ possible states in each time step. The transition rule establishes how the states will change through time based on

the current state of each cell and its immediate neighbors. For one-dimensional CA, the neighborhood size μ is usually written as $\mu = 2R + 1$, where R is the radius. The state α_i of the i_{th} cell of the lattice at time $\tau + 1$ is denoted by:

$$\alpha_i^{\tau+1} = \Delta[\alpha_{i-R}^{\tau}, \ldots, \alpha_i^{\tau}, \ldots, \alpha_{i+R}^{\tau}] \quad (1)$$

where Δ is a transition rule. Note that the state of $\alpha_i^{\tau+1}$ depends only on the states of itself and its neighbors at time τ.

In a binary CA (i.e., two-state), the transition rule Δ is given by a rule table, which lists each possible neighborhood with its output bit, i.e. the updating value of the center cell of the neighborhood. Cells updating usually happens in the following ways: (i) parallel or synchronous, in which all cells of the lattice update their states synchronously at each time step; (ii) sequential or asynchronous, in which only one cell updates its state and this new state is considered in the update of other cells being that the order in which each cell is updated is from the left to the right [5]. Note that when synchronous updating is used, the new states of cells α_i and α_{i+1} can be calculated at the same time step, whereas when asynchronous updating is used, α_i must be calculated in a time step and its new state is employed to update α_{i+1} in the next time step.

2.2 Task Scheduling

Scheduling is an essential task for industries and organizations and also an important subject of many research areas in engineering and computing. In a broad sense, scheduling is a decision-making process that involves resources and tasks to optimize an objective, typically the resultant runtime or makespan [11]. Some applications for scheduling comprehend production scheduling, employees scheduling and computational tasks scheduling.

Considering the multiprocessors scheduling context, the objective is to allocate a set of computational tasks that compose a *parallel application* into architecture nodes. In the problem investigated here, all information about the tasks is known a priori and is named Static Task Scheduling [8]. An optimal solution to an instance of the problem is such that the precedence constraints among tasks are satisfied and the makespan is minimized. According to [7], task scheduling is a NP-Complete problem, even limited to the simplest case: a parallel system with only two processors.

The key concepts adopted here for the representation of the task scheduling problem are described below:

- A parallel application is represented by a directed acyclic graph (DAG) called *program graph*. Fig. 1 shows an example of a program graph called *gauss18*, which represents a set of 18 tasks.
- Computational tasks are represented by nodes (V).
- Precedence constraints between tasks are denoted by edges (E).
- For each node v_i, a cost w_i relative to its runtime is associated.

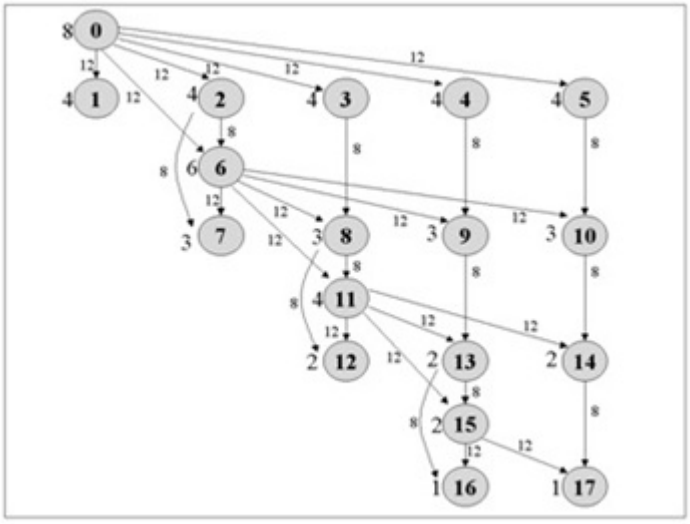

Fig. 1 Example of a program graph with 18 tasks (*gauss18*).

- For each edge $e_{i,j}$, a communication cost $c_{i,j}$ relative to the cost of data transfer from task i to j when running on different processors is associated.
- A task can not be executed unless all its predecessors have completed their executions and all relevant data are available.
- Tasks without predecessors are called starting tasks and tasks without successors are called exit tasks.
- A scheduling policy defines the running order of tasks in each processor. Note that while the scheduler distributes tasks among processors, the scheduling policy orders these tasks within each processor.

Here, the scheduling policy used for all tests was the task with the highest dynamic blevel first. The blevel (bottom level) of a task in a program graph is the highest cost between this task and an exit task of graph, thus the blevel of a task i can be calculated by:

$$bl_i = \begin{cases} w_i, \text{ if } i \text{ is an exit task;} \\ \max_{j \in successors(i)}(bl_j + c_{i,j}) + w_i, \text{ otherwise.} \end{cases} \quad (2)$$

where bl_j denotes the blevel of each successor of i. Blevel of tasks without successors is equal to their respective computational cost (w). For another tasks, blevel is obtained recursively from exit tasks.

The blevel of a task is dynamic when it is calculated considering the allocation of the tasks in processors, and the communication cost is considered only when tasks are distributed in different processors [2].

2.3 CA-based scheduler: concepts and related works

Previous CA-based scheduler models assume that each cell of the lattice is associated with a computational task of the target program graph [14]. Therefore, if a set of tasks has cardinality x, CA lattice has $\eta = x$ cells. Furthermore, given an architecture consisting of p processors, CA will have $\kappa = p$ possible states. Assuming a system with two processors (P_0 and P_1), each cell can take value 0, indicating that the corresponding task is allocated on processor P_0, or value 1 (the task is allocated on P_1). Fig. 2 shows an example of a problem modeling using the CA approach proposed in [14] and used in subsequent works [15,16,2,1,3]. First, we have a program graph (with four tasks) and a multiprocessor system (with two processors). Based on this information, $\eta = 4$ and $\kappa = 2$. The algorithm makes an initial allocation and represents it as the initial configuration of CA lattice. Starting from this initial lattice, a transition rule Δ is applied by τ time steps. The final lattice is then associated with the final allocation of the tasks in the processors. Finally, a scheduling policy is applied to the final allocation and the makespan is obtained.

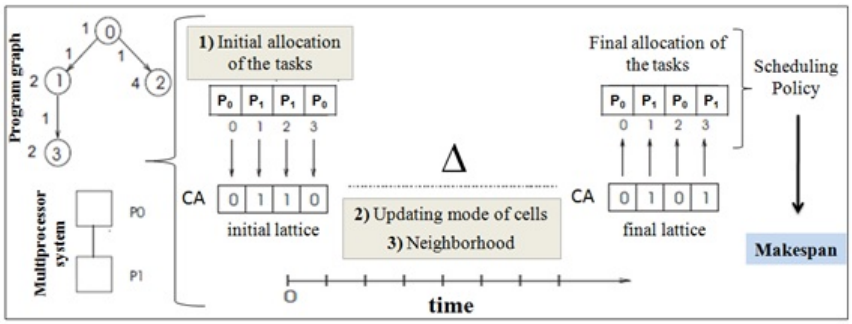

Fig. 2 General scheme of CA-based scheduling.

Fig. 2 shows that the transition rule Δ plays a key role in CA-based scheduling. It is necessary to find transition rules with computational ability for solving task scheduling. A genetic algorithm (GA) was employed in [14] to discover CA rules. Most subsequent works also used a simple genetic algorithm to discover rules able to schedule a specific target graph. These CA-based models operate in two stages: learning and reuse.

In the learning phase, a GA or another evolutionary algorithm is used to search for rules able to evolve the lattice to optimal (or sub-optimal) allocations of a given program graph, starting from some initial configurations. In [14], the GA population (ga_P) is initially formed by CA transition rules randomly generated (individuals). The fitness function is calculated in each GA generation by: (i) a set of initial lattices Γ representing allocations of tasks in processors is generated; (ii) temporal evolution of each lattice Γ by each transition rule Δ in ga_P for τ time steps; (iii) allocations obtained in time τ

are ordered in each processor using a scheduling policy obtaining makespan associated to each pair (Δ, Γ); (iv) rule fitness is given by the average of makespan calculated starting from each lattice of Γ. The best rule shows the smallest makespan average. After computing the fitness, genetic operators as selection, crossover and mutation are applied to the population generating new transition rules. At the end of each generation, a re-insertion criterion defines which individuals remain for the next generation. Subsequent works employed different evolutionary algorithms to the learning phase as simple genetic algorithms [15,2,1,3], coevolutionary genetic algorithms [14,10] and joint evolution genetic algorithms [16].

CA rules stored after the learning phase are expected to be able to make a good scheduling for other program graphs. This stage is named reuse phase; the rules learned for a specific program graph are applied to new instances of program graphs. However, recent works have pointed that rules obtained in the learning phase do not have the generalization ability to be applied to other instances as expected [3]. These works focus on obtaining a better capacity to apply rules learned to other program graphs.

As highlighted in Fig. 2, some variations in the major steps of CA-based scheduling have been largely investigated in the literature:

1. *Lattice initializations* - there are different ways to establish the initial configuration of the lattice, which corresponds to the initial allocation of the tasks over the nodes of the multiprocessor architecture. They can be obtained randomly [14,15,16,2], through fixed and very easy strategies [1] or considering the scheduling performed by a very simple heuristic [3], as the initialization strategy investigated here.
2. *Updating mode of CA cells* - different approaches were also investigated, being that sequential, sequential-randomly and parallel have been used. Although in the first published models [14,15,16] the sequential updating returned the best results, parallel updating is desirable [14] since the inherit parallelism of CA implementations can only be exploited in this updating mode. Recent investigations have focused on synchronous models [2,1,3].
3. *Neighborhood models* - these models are responsible for capturing the relations between tasks expressed in the program graph. Previous neighborhood models are linear and non-linear. The advantage of linear neighborhood is that it is simple and easy to adapt to an arbitrary number of processors [15]. However, it does not capture the actual relations between tasks because it is based only on the order number of the tasks (the neighborhood of a cell α_i is chosen by the position in the lattice, not by the relations between tasks). On the other hand, non-linear neighborhoods can express very complex relations between tasks, such as precedence constraints or dependences of the same task (two tasks that are preceded by a same task). The nonlinear models previously investigated in the literature were: selected and totalistic [14,16,10]. However, they are computationally intensive and difficult to adapt to architectures using more than two processors. In this paper, we propose a new neighborhood model, named

here pseudo-linear, which keeps the same simplicity of linear model, but expresses more information about the dependences in the program graph.

3 New approaches to CA-based scheduling

There are many important elements in the CA-based scheduling. In this paper, we explore two innovative approaches. The first algorithm is called Synchronous Cellular Automata-based Scheduler with initialization Heuristic (SCAS-H). It was preliminarily presented in [3]. Now, we show a general description of it and a more extensive experimental analysis. The second approach is a new neighborhood model called pseudo-linear, able to capture the proximity and relations strength among tasks in a program graph.

This section is organized as follows: Subsect. 3.1 introduces an analysis of previous models (CAS and SCAS); Subsect. 3.2 describes SCAS-H technique; and Subsect. 3.3 presents the new neighborhood framework.

3.1 CAS and SCAS: analysis of previous models

[14] presents a CA-based scheduler that operates in two modes: learning and operation. In the learning mode, a genetic algorithm (GA) is used to search for rules able to evolve the lattice to optimal (or sub-optimal) allocations of a given program graph. CA evolution starts from random initial configurations. In the operation mode is expected that, for any initial allocation of tasks, the CA rules stored after learning phase are able to evolve the lattice until a configuration which represents an optimal allocation, minimizing the makespan. The rules obtained in the learning phase are also expected to be used in the scheduling of other graphs (reuse phase). The neighborhood model employed in [15] is linear and both updating modes of cells - sequential and parallel - were investigated. Moreover, the results obtained with the sequential updating were much better than using parallel. We named this model CAS (CA-based scheduler).

In the first CA-based scheduler models the synchronous updating mode of cells was discarded because it returned the worst results [15]. On the other hand, the large capacity of parallelism inherent to CA is lost if the asynchronous updating of cells is adopted [14,2]. A CA-based scheduler model using synchronous updating of cells was introduced in [2]. This model, named SCAS (Synchronous Cellular Automata-based Scheduler), also employs linear neighborhood, but unlike [15], the strategy in GA is not elitist. In addition, the boundary condition used in the CA lattice is different from the null condition employed in [15]: cells to the right of the last cell are considered in state 1. The results using SCAS showed its good performance in comparison to CAS: its results overcame CAS model with synchronous updating and they are compatible with CAS model using sequential updating [2].

SCAS was used as the basis of the new models investigated in the present work. That is, all of them employ synchronous updating and non-elitist strategy in the genetic algorithm used in the learning phase.

One of the major motivations to build new CA-based scheduler models is the great difficulty observed in some related works when the number of processors increases [15], while investigations performed in other works consider only multiprocessor system with $\kappa = 2$ processors [14,16,2,10,3]. An attempt to increase the number of processors was presented in [15], showing CAS model was not able to deal with the complexity due to the increment in the number of processors. Another motivation to build new CA-based scheduler models was the undesirable results provided by the reuse of the rules found for a given program graph to other unseen instances.

We implemented the CAS model with sequential updating as described in [15] and the previous SCAS model as described in [2] to evaluate the reuse of rules evolved for *gauss18* in the learning phase to solve different program graphs. We also implemented a scheduling algorithm based on the meta-heuristics simulated annealing, named here SA, to evaluate the quality of solutions given by the reuse of *gauss18*-learned rules. The results of reusing *gauss18* rules (evolved previously using CAS and SCAS models) are not reasonable when compared with the reference values given by SA. We also evaluated the previous models CAS and SCAS when using more than $\kappa = 2$ processors in the learning phase and compared their results with those obtained by SA. Both models (CAS and SCAS) presented limitations when the number of processors has increased. Sect. 4 shows and discusses the results of these experiments in detail.

We believe that an important cause to this undesirable performance is related to the process used to initialize CA lattices to start scheduling in CAS and SCAS models. The way how the initial configuration of the CA lattice is defined reflects the type of transition rule ability the GA is searching for. In previous studies [14,15,16,2,10], the scheduler model focused on the capacity of a transition rule to evolve any random initial lattice to a configuration that represents the optimal allocation of tasks. During the learning phase, each rule is evaluated according to its performance in scheduling a set of initial lattices Γ. Besides, in the operation phase, the quality of any learned rule is measured using it to schedule a new set of random lattices.

However, the search for this independence to the initial lattice makes the rules search complex and computationally intensive, embarrassing GA convergence. The capacity of a rule to schedule other instances in the reuse phase is a more relevant generalization ability than the capacity of a rule to perform schedule starting from any initial lattice. Therefore, a new way to start the scheduling from a specific initial condition generated by a simple heuristic is investigated here. This modification leads to the first model evaluated in the present work: SCAS-H.

3.2 SCAS-H: Synchronous Cellular Automata-based Scheduler initialized by Heuristic

The major modification of SCAS-H in comparison with the previous SCAS model refers to the way the CA lattice is initialized to perform the schedule, which reflects in the process of evaluation of the GA rule population. The main steps in the CA evolution used in SCAS-H are (i) application of a deterministic construction heuristic chosen a priori to obtain an initial allocation, which defines the initial configuration of the lattice; (ii) temporal evolution of the lattice using each rule transition Δ in ga_P for τ time steps; (iii) the final lattice in time τ defines the final allocation of tasks which are ordered in each processor using a scheduling policy obtaining makespan associated to each rule Δ; (iv) rule fitness is equal to makespan obtained using it. The best rule in ga_P shows the smallest makespan. In fact, step (i) is performed only once during GA run, because the allocation performed by the deterministic construction heuristic is unique and is used in all evaluations.

Heuristic methods have been common in the literature to approximately solve task scheduling in a reasonable time [8]. These methods build a single response to a given input in each step of scheduling and they are known as construction heuristics. They are characterized by low computational complexity and utilization of attributes calculated directly from the program graph to perform the scheduling. HLFET (Highest Level First with Estimated Time) [8], a widely known construction heuristic, was slightly modified to generate the initial configuration of the lattice. We built a deterministic heuristic called DHLFET, which it is HLFET without its random choices. Therefore, in case of two tasks with same sl, the task with the smallest order number is chosen. Alg. 1 shows DHLFET steps. The static level attribute (sl) computes the largest path from each task to an exit task without considering communication costs in the program graph.

Algorithm 1 DHLFET Heuristic

1: Compute the *sl* (static level attribute) for each task
2: Make a *ReadyList* in a descending order of *sl*. First, *ReadyList* includes only starting tasks. Ties are broken by chose the task with smallest order number
3: **while** all tasks are not scheduled **do**
4: Schedule the head task of *ReadyList* for the processor that permits the earliest execution using non-insertion approach
5: Update *ReadyList* including the new ready tasks
6: **end while**

SCAS-H works in two stages: learning and reuse. Algorithm 2 defines the major steps of SCAS-H in the learning mode, where ga_G is the number of generations (and the stopping criterion), ga_P is the size of population, ga_C is the number of generated individuals in crossover given by ga_{PC} (crossover rate), ga_M is the mutation rate, selection is given by a simple tournament ga_T

and IC_H is the initial configuration of the lattice obtained by the heuristic (DHLFET).

Algorithm 2 SCAS-H Learning Mode

1: Generate a random population of ga_P rules
2: Generate initial configuration of lattice with DHLFET (IC_H)
3: Compute the fitness of the ga_P rules
4: **while** ($!finish$) **do**
5: Selecting pairs of rules in ga_P using simple tournament ($ga_T = 2$) to generate ga_C rules
6: Applying the single-point crossover in pairs selected
7: Submit ga_C rules to mutation ga_M
8: Compute the fitness of the ga_C rules
9: Sorting $ga_P + ga_C$ based on fitness and choose the ga_P best rules for next generation
10: **end while**
11: ga_P is stored in rules database

In reuse mode, the CA is equipped with a set of learned rules and SCAS-H receives a new program graph to schedule. The same heuristic used in the learning mode is used in the reuse mode to make the initial allocation associated to the new graph. Then, CA steps are executed for each rule Δ, returning the scheduling with the smallest makespan.

Results of experiments comparing SCAS-H with the previous models CAS [15] and SCAS [2] are provided in Subsect. 4.1. They show the improvement obtained with the introduction of the new strategy to initialize the CA lattice using a construction heuristic with significant impact on the learning phase when using more than two processors. However, subsequent experiments show that this improvement was not so emphatic in the reuse phase, where scheduler performance decays when architectures with $\kappa = 3$ or $\kappa = 4$ processors are used.

It was possible to conclude that this limitation to manipulate more than two processors in the reuse phase is partially related to the linear neighborhood employed in previous models and in SCAS-H. Based on this observation, a new neighborhood model was proposed and is named here pseudo-linear. It preserves the simple structure of linear neighborhood, but the neighbors relations are defined by the proximity and relative importance of the tasks within the program graph.

3.3 Pseudo-linear neighborhood for CA-based scheduling

Neighborhood is a crucial element to CA transition rules because it can express relations among tasks in program graph. In this context, two approaches have been largely explored in related works: linear [15, 2, 1, 3] and nonlinear neighborhoods [14, 16]. Linear neighborhood is very simple to implement because it uses only the order number of the tasks and a radius R to define the

neighborhood. It can be easily adapted so that an arbitrary number of processors can be used. However, it is not adequate to identify the relationships among tasks in the program graph. On the other hand, two nonlinear neighborhoods have been investigated in the literature: selected [14,16] and totalistic [14]. Although they can capture the relations among tasks from precedence constraints and attributes of the program graph, they are very complex to implement and limited to multiprocessor systems with only two processors.

In this context, a new neighborhood model named pseudo-linear is proposed here. As the linear neighborhood, the new model uses radius R, but it is able to capture relations among computational tasks in the program graph. Furthermore, it can be adapted to multiprocessor systems with an arbitrary number of processors. Pseudo-linear is a simple neighborhood based on two important concepts. First, it considers only direct relationships between tasks in the graph: precedence constraint. For example, v_i can be a neighbor of v_j if and only if there is a precedence constraint $e_{i,j}$ or $e_{j,i}$ in the program graph. This means that pseudo-linear considers only the predecessors or successors of a given task to define its neighborhood. In addition, for each task and edge in the graph is associated a cost, which is important to detect the strength of the relationship among one task and its predecessors or successors. In this investigation, pseudo-linear neighborhood employs two well-known attributes in task scheduling: bottom level of a task (or *blevel*) and top level of a task (or *tlevel*) [8]. Note that pseudo-linear neighborhood is a framework and other attributes can be considered.

A more detailed explanation about the pseudo-linear neighborhood is given as follows. Through a radius R, we can determine the next state of cell σ_i according to (3):

$$\sigma_i^{\tau+1} = \Delta[\sigma^{\tau}_{blevel(R)}, \ldots, \sigma^{\tau}_{blevel(1)}, \sigma_i^{\tau}, \sigma^{\tau}_{tlevel(1)}, \ldots, \sigma^{\tau}_{tlevel(R)}] \tag{3}$$

where function $blevel(.)$ returns the task associated with cell σ_i by a precedence relation (successor or predecessor). Initially, the set of successors and predecessors of task σ_i are identified in a list of related tasks. Thus, the list of tasks is ordered by *blevel* attribute in a descending order and function $blevel(x)$ returns the x^{th} task in this ordered list. A similar procedure is performed for function $tlevel(x)$, which returns the x^{th} related task considering the *tlevel* descending order. For each attribute (*blevel* and *tlevel*), using a neighborhood with radius R, only the first R tasks in each list are used as neighbors. Note that when two or more tasks have the same value for some attribute, (i) *blevel* list orders these tasks according to the highest order number, whereas (ii) *tlevel* list orders these tasks according to the lowest *alap* (As Late As Possible start time attribute [8]). If two or more tasks have the same *alap*, *tlevel* list orders these tasks according to the lowest order number. Furthermore, there is only one special case in pseudo-linear neighborhood: when the number of neighbors is smaller than R. In this case, the lists come back to head task and continue until they complete R neighbors tasks.

Fig. 3 illustrates pseudo-linear neighborhood. Fig. 3(a) shows the direct neighbors of task 11 in *gauss18* program graph (Fig. 1). Figs. 3(b) and 3(c)

display, respectively, the resultant neighborhood using linear and pseudo-linear neighborhood for this task. Note that linear neighborhood considers only the position of the tasks in the lattice, which explains why tasks 9, 10, 12 and 13 are considered neighbors of task 11 (for $R = 2$). Therefore, for every program graph, such as *rand30*, *rand40* and *rand50*, the neighbors of task 11 are the same, regardless of the relationship among the tasks. On the other hand, pseudo-linear considers tasks 8, 6, 15 and 13 neighbors of 11 because it is able to capture proximity and strength of the relations among tasks in a program graph.

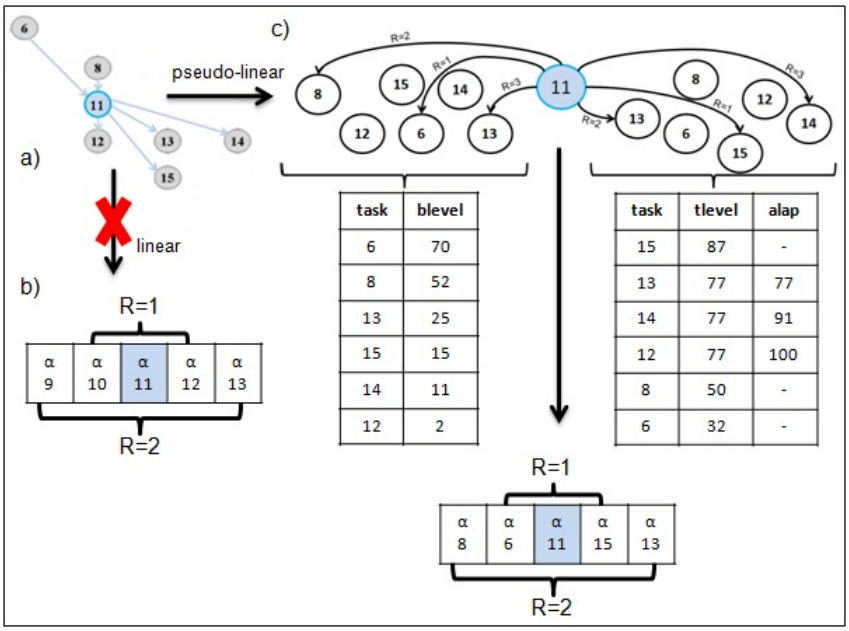

Fig. 3 Examples of neighborhoods of task 11 in *gauss18*: (a) precedence constraints related to task 11; (b) linear neighborhood for task 11; (c) pseudo-linear neighborhood for task 11.

The new model implemented using pseudo-linear neighborhood was named SCAS-HP. It basically uses similar steps of SCAS-H, except by the neighborhood model adopted for the transition rules. Thus, the major modification is related to step 3 of Alg. 2 because the pseudo-linear neighborhood is adopted instead of the linear one during the CA evolution. Furthermore, SCAS-HP continues working in two stages (learning and reuse) and Alg. 2 can be also used to define the major steps of SCAS-HP in the learning mode.

Comparative experiments using SCAS-H and SCAS-HP are reported in Section 4.2. They show that the adoption of the new neighborhood model increases the generalization ability of the rules in the reuse phase for architectures with more than two processors.

4 Experimental results and discussion

This section presents and discusses some results of computer simulations carried out in order to extensively evaluate SCAS-H and SCAS-HP models. Specifically, Subsects. 4.1 and 4.2 provide detailed simulations in the learning phase and supply the performance of the proposed framework in the reuse phase under real-world conditions, respectively, for SCAS-H and SCAS-HP. A general explanation about the experiments is given below.

- **Program graphs**
  In all simulations we used the most difficult program graph found in the literature, *gauss18*. We also adopted a modified version of DAG generation [9] to obtain randomly program graphs *rand30*, *rand40* and *rand50* with a high nonlinearity similar to *gauss18*. These program graphs have, respectively, 30, 40 and 50 tasks.
- **Algorithms and parameters**
  Other techniques, such as CAS [15], SCAS [2] and Simulated Annealing [12] were applied to these program graphs and the results were compared with those of proposed models.
  The parameters of the evolutionary algorithm in all CA-based schedulers were size of population $ga_P = 200$, simple tournament $ga_T = 2$, crossover rate $ga_{PC} = 100\%$, mutation rate $ga_M = 3\%$ and number of generations $ga_G = 200$.
  After some empirical tests, SA parameters were given by:

$$sa_{temp} = 100 * 0.9995 sa_{time} \tag{4}$$

  where sa_{temp} represents the temperature mapping in function of time sa_{time}. Other parameters were $\epsilon = 1 * 10^{-9}$ and lateral moves $sa_{moves} = \eta/6$.
- **Analysis of results**
  Twenty runs were performed for each experiment. Statistical analysis was employed to evaluate the techniques. When the samples follow a normal distribution, t-test is used, otherwise we applied Mann-Whitney test. For the hypothesis test, we adopted a significance level of 5%, which corresponds to a confidence level of 95%. Analysis involving best makespan ("Mk"), averages and standard deviation were also conducted.

4.1 SCAS-H experimental analysis

This subsection reports on a comparative study about the performance of SCAS-H, previous models of CA-based scheduler and SA in multiprocessor systems with a different number of processors. We divided the analysis into two topics: experimental results in learning phase and reuse phase.

4.1.1 Learning phase

Tab. 1 shows the results of the learning phase in CA-based approaches and SA for $\kappa = 2$, $\kappa = 3$ and $\kappa = 4$ number of processors. Column "Mk" represents the best makespan value in all executions. Note that in this column, the results of the reproduction of CAS for sequential updating of cells (CAS) [15] and SCAS [2] (which uses synchronous updating of cells) do not ensure an exact makespan value because these algorithms use a Γ set of random initial configurations to drive the learning and reuse phases. So, the makespan value of a transition rule Δ is an average of the results obtained by Δ on the Γ set [3]. On the other hand, SCAS-H does not display this characteristic because it always starts from the same initial configuration obtained by the construction heuristic. Column "H" denotes the results of the statistical tests when comparing SCAS-H with other algorithms. In other words, "<" represents that SCAS-H obtain better results than the considered algorithm, ">" represents the opposite, and "=" states that there is not statistical evidence about the better algorithm (null hypothesis).

Table 1 Comparison among SCAS-H learning phase and CAS, SCAS and SA techniques in multiprocessor systems with two ($\kappa = 2$), three ($\kappa = 3$) and four processors ($\kappa = 4$).

Learning phase - CA parameters:
$\kappa = 2$ ($R = 3$), $\kappa = \{3, 4\}$ ($R = 1$), $\tau = 50$

		SCAS-H	**CAS**		**SCAS**		**SA**	
$\mathbf{P_G}$	κ	Mk	Mk	H	Mk	H	Mk	H
gauss18	2	44	44	<	44	<	44	<
	3	44	52	<	52	<	44	<
	4	44	51.7	<	50.8	<	44	<
rand30	2	1222	1239	<	1225.84	<	1222	<
	3	853	963	<	1012.86	<	970	<
	4	828	902.72	<	976.4	<	853	<
rand40	2	983	1006	<	996.52	<	997	<
	3	694	806.94	<	796.6	<	794	<
	4	607	681	<	725.4	<	684	<
rand50	2	628	659.04	<	661.04	<	664	<
	3	532	640.96	<	655.12	<	624	<
	4	524	620	<	642.64	<	600	<

Discussion for $\kappa = 2$ results: For $\kappa = 2$, SCAS-H shows the best results in comparison with previous CA-based approaches in the learning stage. Statistical tests prove this. Although average and standard deviation results are not shown in Tab. 1 for clarity, it was possible to observe that standard deviation in SCAS-H results is much smaller when compared to other techniques. This shows learning phase in SCAS-H is very robust in $\kappa = 2$ multiprocessor systems.

Discussion for $\kappa = 3$ results: There are significance statistical evidence that SCAS-H results were better than those obtained by other techniques. Note that for $\kappa = 3$, the difference among SCAS-H results and other algorithms is

greater than considering $\kappa = 2$. Moreover, for this number of processors, previous CA-based approaches showed worse results in *gauss18* program graph. Both approaches and SA also found the worst results on randomly generated program graphs. On the other hand, SCAS-H was able to extract rules that provides good scheduling to the program graphs.

Discussion for $\kappa = 4$ results: As in the experiments discussed above ($\kappa = 2$ and $\kappa = 3$), previous CA-based approaches showed the worst results to schedule *gauss18* and random program graphs. SA found worse results than SCAS-H in random program graphs. Once again, SCAS-H presented the best performance according to statistical tests.

4.1.2 Reuse phase

Tab. 2 shows the best makespan after applying the extracted rules learned for *gauss18* program graph for $\kappa = 2$ processors on distinct program graphs: *rand30*, *rand40* and *rand50*. Comparisons among SCAS-H performance and other CA-based algorithms showed the best results were obtained by SCAS-H (one can clearly see a great difference). Comparing SCAS-H with SA, the results are close. However, it is important notice that SA executes all steps of its search for each instance whereas SCAS-H uses only the learned rules extracted from another program graph. As a general conclusion, we could notice that there was space to extract better performance from learned CA-rules. So, we started to investigate new models of neighborhood to better represent the relationship between tasks.

Table 2 Learned *gauss18* transition rules of CAS, SCAS and SCAS-H applied to the reuse phase considering multiprocessor systems with $\kappa = 2$.

***gauss18* Reuse phase**			
Alg.	**rand30**	**rand40**	**rand50**
SCAS-H	1233	1000	**656**
CAS	1336.61	1136.55	730.7
SCAS	1743.25	1268.29	837.22
SA	**1222**	**997**	664

4.2 Pseudo-linear neighborhood experimental analysis

Many CA-based approaches have used either linear neighborhood [15, 2, 1, 3] or nonlinear neighborhood [14, 16, 10]. However, these two approaches have their limitations: the former uses only the position in the lattice to determine the neighborhood of a task whereas the latter is complex and can be applied only to multiprocessor systems with two processors. To deal with these drawbacks, we

propose SCAS-HP, a CA-based scheduling that uses the pseudo-linear neighborhood. This new neighborhood model offers CA a simple structure to capture the dependence and relationship strength among tasks in the program graph and can be applied with an arbitrary number of processors.

This subsection describes experiments performed considering the environment with the new neighborhood model. Subsect. 4.2.1 provides results of SCAS-HP in the learning phase compared with those obtained by the powerful CA-based scheduler showed previously, SCAS-H. Subsect. 4.2.2 shows the reuse phase analysis for these algorithms. In addition, SCAS-HP also is compared with SA in these subsections.

4.2.1 Learning phase

Tab. 3 provides a detailed analysis of the results of SCAS-HP, SCAS-H and SA considering multiprocessor systems with $\kappa = 2$, $\kappa = 3$ and $\kappa = 4$. Note that statistical tests show a comparison among SCAS-HP and other approaches. For instance, column "H" in line "SCAS-H" represents the results of the statistical tests between SCAS-HP and SCAS-H ("<" indicates there is significance statistical evidence than SCAS-HP presents better performance; ">"indicates SCAS-H has better results; "=" indicates there is not statistical evidence than a algorithm is better than other).

Table 3 Comparison among SCAS-HP learning phase and SCAS-H and SA techniques in multiprocessor systems with two ($\kappa = 2$), three ($\kappa = 3$) and four processors ($\kappa = 4$).

Learning phase - CA parameters:
$\kappa = 2$ ($R = 3$), $\kappa = \{3, 4\}$ ($R = 1$), $\tau = \eta * 3$

		SCAS-HP	**SCAS-H**		**SA**	
$\mathbf{P_G}$	κ	Mk	Mk	H	Mk	H
gauss18	2	44	44	=	44	<
	3	44	44	>	44	<
	4	44	44	=	44	<
rand30	2	1222	1222	=	1222	<
	3	836	851	<	970	<
	4	755	822	<	853	<
rand40	2	983	983	=	997	<
	3	684	695	<	794	<
	4	564	613	<	684	<
rand50	2	624	628	<	664	<
	3	544	528	>	624	<
	4	504	540	<	600	<

Discussion for $\kappa = 2$ results: These results show the power of the learning stage in SCAS-H and SCAS-HP. Only considering *rand50* program graph, there is significance statistical evidence that SCAS-HP results are better than those of SCAS-H. Comparison between SCAS-HP and SA shows a reasonable difference between makespan values on random program graphs. Furthermore,

there is statistical evidence that SCAS-HP results are better than SA on all program graphs.

Discussion for $\kappa = 3$ results: Again, the results of SCAS-HP are better than those found by SA according to stastistical tests. However, SCAS-H is better than SCAS-HP on two program graphs (*gauss18* and *rand50*), whereas SCAS-HP is better on the other two program graphs (*rand30* and *rand40*).

Discussion for $\kappa = 4$ results: An interesting result in this table is provided by SCAS-HP, which found the optimal makespan for *gauss18* program graph in all twenty runs. SCAS-H was near this result on *gauss18*. However, in other program graphs, there is statistical evidence that SCAS-HP results are better than those found by SCAS-H. Furthermore, there are great differences between the makespan and the averages obtained by these two models. In addition, the table shows SA presented the worst results.

4.2.2 Reuse phase

Aiming to evaluate CA-based scheduling algorithms under real conditions, we took the extracted rules in the learning phase of *gauss18* program graph considering $\kappa = 2$, $\kappa = 3$ and $\kappa = 4$ number of processors. These rules are applied to distinct program graphs considering the same κ value and the makespan is obtained. For example, rules extracted from *gauss18* learning phase considering $\kappa = 3$ processors are used to schedule *rand30*, *rand40* and *rand50* program graph on $\kappa = 3$ processors.

Tab. 4 shows the reuse of rules extracted from *gauss18* program graph for SCAS-H and SCAS-HP. Note that in 8 out of 9 cases, the results of SCAS-HP are better than those obtained by SCAS-H. When comparing SCAS-HP with SA, the former yelded the best results in 6 out of 9 cases. Furthermore, remember that SA needs to perform its search process again for each scheduling whereas CA-based models only reuse the acquired knowldege.

Table 4 Learned *gauss18* transition rules of SCAS-HP and SCAS-H applied to the reuse phase considering multiprocessor systems with different numbers of processors ($\kappa = 2$, $\kappa = 3$ and $\kappa = 4$).

***gauss18* Reuse phase**									
	rand30			**rand40**			**rand50**		
Alg.	2	3	4	2	3	4	2	3	4
SCAS-HP	1232	**938**	857	**990**	**733**	**656**	**644**	644	**596**
SCAS-H	1244	1122	959	1007	848	803	660	628	688
SA	**1222**	970	**853**	997	794	684	664	**624**	600

5 Conclusions

This paper presents two new approaches to CA-based scheduling: (i) employment of a construction heuristic to initialize CA lattice evolution, and (ii) a

new neighborhood model - named pseudo-linear - able to capture the dependence and relations strength among the tasks of the program graphs in a very simple way. The first approach leads to the first scheduler model investigated here named SCAS-H and the second one enables to refine SCAS-H and to propose the second scheduler model named SCAS-HP.

As construction heuristic, we proposed DHLFET. It is a slight modification of the well-known HLFET heuristic [8], which is a very simple and computationally efficient heuristic commonly used in scheduling task. For the pseudo-linear neighborhood, we employed the bottom level (*blevel*) and top level (*tlevel*) attributes, since these measures can do a good characterization of the relations among tasks considering scheduling context. Several experiments were conducted in the learning and reuse phases, so that we could better assess the performance of the proposed techniques. Parallel program graphs found in the literature and others randomly generated were used in the experiments. SCAS-H using DHLFET in the lattice initialization and SCAS-HP using DHLFET and the pseudo-linear neighborhood were extensively evaluated in multiprocessor systems with two, three and four processors. Their performances were compared with those of previous CA-based schedulers and a simple scheduler based on Simulated Annealing meta-heuristic.

First experiments showed SCAS-H overcame previous CA-based models proposed in [15] and [2] showing that the employment of a simple heuristic to initialize the CA lattice evolution can be more efficient than the usage of a set of random lattices to guide genetic search in the learning phase. This improvement was highlighted when the number of processors on the multiprocessor architecture was raised from 2 to 3 and 4 processors. Comparing SCAS-H with a second scheduler based on simulated annealing (SA) it was possible to notice SCAS-H were consistently better than SA no matter the number of processors employed (2, 3 or 4) when considering the learning phase. That is, when both models SCAS-H and SA employ a search taking the target graph in account, the CA-based model returns a better performance. However, when we tried to reuse SCAS-H rules previously evolved for a target graph to new program graphs unseen during the evolutionary search, makespam values were worse than the results obtained by SA. Although it must taken account that SA needs to perform the search again for each program graph, while CA rules are reused without search process, the main goal of CA-based schedulers is the possibility to have good results in reuse phase. Therefore, we concluded we have space to improve SCAS-H reusing results.

The second series of experiments showed that SCAS-HP is also consistently better than SA scheduler during learning phase, as observed for SCAS-H. Comparing the performance of both CA-based models investigated here during learning phase, SCAS-HP returned results at least as good as SCAS-H, with some significant improvement in some scenarios showing that the new pseudo-linear neighborhood is good to express program graph relations. Considering 4 processors in the architecture, SCAS-HP improves SCAS-H with a significant difference, as showed by statistical tests. Besides, in reuse phase, results obtained employing SCAS-HP rules evolved for *gauss18* to other unseen

program graphs showed a significant advantage over SCAS-H in 8 of 9 evaluated scenarios (3 unseen graphs, with 2, 3 and 4 processors). Due to SCAS-HP performance it seems that pseudo-linear neighborhood is better than the linear model to capture task relations expressed in the program graph and CA rules evolved using this new model have a better generalization ability, which is a desirable characteristic in CA-based schedulers [10]. SA scheduler results are more close to SCAS-HP reuse results; however, we must highlight that SA do a new search for each graph and even this, SCAS-HP was better than SA in 6 of the 9 scenarios.

As a general conclusion, the performances of the proposed approaches are better than those obtained by other CA-based algorithms as in the learning stage as in the reuse phase. Finally, experimental analysis also drive us to conclude that the combined employment of both techniques make the search for CA transition rules during learning more robust and leads to a significant gain when considering the reuse of them on real-world conditions.

As future work, SCAS-HP will be extended by the investigation of new heuristics to initialize CA lattices. Other attributes, such as *alap* (As Late As Possible), *asap* (As Soon As Possible) and *cp* (Critical Path), will also be tested to define the pseudo-linear neighborhood. The inspection of the minority of scenarios in which SCAS-HP did not return a good result revealed that the change performed in the initialization of lattices, where we use only one configuration to learn the rules instead of a set of configurations, together with the application of a more complex neighborhood can provoke some undesirable unstable behavior, as chaotic dynamics. We are working on an approach to guide genetic search to avoid such unstable rules. Initial results are very promising and they must be divulged next soon.

Acknowledgements

This work has been supported by the National Counsel of Technological and Scientific Development - CNPq (process 134278/2010-0) - of the Brazilian Government. Murillo Guimarães Carneiro thanks also to São Paulo Research Foundation (FAPESP) and Gina Maira Barbosa de Oliveira is grateful to Minas Gerais Research Foundation (FAPEMIG).

References

1. Carneiro, M.G., Oliveira, G.M.: SCAS-IS: Knowledge extraction and reuse in multiprocessor task scheduling based on cellular automata. In: Proceedings of Brazilian Symposium on Neural Networks (SBRN), pp. 142–147 (2012)
2. Carneiro, M.G., Oliveira, G.M.B.: Cellular automata-based model with synchronous updating for task static scheduling. In: Proceedings of 17th International Workshop on Cellular Automata and Discrete Complex System, pp. 263–272 (2011)
3. Carneiro, M.G., Oliveira, G.M.B.: SCAS-H: Synchronous cellular automata-based scheduler with initialization heuristic to task scheduling. In: Proceedings of 18th International Workshop on Cellular Automata and Discrete Complex System, pp. 1–10 (2012)

4. Dennunzio, A.: From one-dimensional to two-dimensional cellular automata. Fundam. Inform. **115**(1), 87–105 (2012)
5. Dennunzio, A., Formenti, E., Manzoni, L.: Computing issues of asynchronous CA. Fundam. Inform. **120**(2), 165–180 (2012)
6. Farina, F., Dennunzio, A.: A predator-prey cellular automaton with parasitic interactions and environmental effects. Fundam. Inf. **83**(4), 337–353 (2008)
7. Garey, M.R., Johnson, D.S.: Computers and Interactability. A Guide to the Theory of NPCompleteness. Freemann And Company (1979)
8. Kwok, Y.K., Ahmad, I.: Benchmarking and comparison of the task graph scheduling algorithms. J. of Parallel and Distributed Computing **59**(3), 381–422 (1999)
9. DAG generation program: http://www.loria.fr/∼suter/dags.html (2011)
10. Oliveira, G.M., Vidica, P.M.: A coevolutionary approach to cellular automata-based task scheduling. In: G.C. Sirakoulis, S. Bandini (eds.) Cellular Automata, *Lecture Notes in Computer Science*, vol. 7495, pp. 111–120. Springer Berlin Heidelberg (2012)
11. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems, third edn. Springer Science (2008)
12. Russell, S., Norvig, P.: Artificial intelligence: a modern approach, third edn. Prentice Hall series in artificial intelligence. Prentice Hall (2010)
13. Sarkar, P.: A brief history of cellular automata. ACM Comp. Surveys **32**(1), 80–107 (2000)
14. Seredynski, F., Zomaya, A.Y.: Sequential and parallel cellular automata-based scheduling algorithms. IEEE Trans. Parallel and Distributed Systems **13**(10), 1009–1022 (2002)
15. Swiecicka, A., Seredynski, F., Zomaya, A.Y.: Multiprocessor scheduling and rescheduling with use of cellular automata and artificial immune system support. IEEE Trans. on Parallel and Distributed Systems **17**(3), 253–262 (2006)
16. Vidica, P.M., Oliveira, G.M.B.: Cellular automata-based scheduling: A new approach to improve generalization ability of evolved rules. pp. 18–23 (2006)
17. Weinert, W.R., Benitez, C., Lopes, H.S., Lima, C.R.E.: Simulation of the dynamic behavior of one-dimensional cellular automata using reconfigurable computing. In: Proceedings of the 3rd international conference on Reconfigurable computing: architectures, tools and applications, ARC'07, pp. 385–390. Springer-Verlag, Berlin, Heidelberg (2007)
18. Wolfram, S.: Cellular automata. Los Alamos Science (1983)
19. Wolfram, S.: Universality and complexity in cellular automata. Physica D: Nonlinear Phenomena **10**, 1 – 35 (1984)
20. Wolfram, S.: Complex systems theory. In: Emerging Syntheses in Science: Proceedings of theFounding Workshops of the Santa Fe Institute, pp. 183–189. Addison-Wesley (1988)
21. Wolfram, S.: Cellular Automata and Complexity. Addison-Wesley (1994)
22. Wolfram, S.: A new kind of science. Wolfram Media, Inc., Champaign, IL (2002)
23. Wolfran, S.: Cryptography with cellular automata. Advances in Cryptology: Crypto '85 Proceedings **218**, 429–432 (1986)