Coarse-Grained Parallelization of Cellular-Automata Simulation Algorithms [⋆]

Olga Bandman

Supercomputer Software Department
ICM&MG, Siberian Branch, Russian Academy of Sciences
Pr. Lavrentieva, 6, Novosibirsk, 630090, Russia
E-mail: bandman@ssd.sscc.ru

Abstract. Simulating spatial dynamics in physics by Cellular Automata (CA) requires very large computation power, and, hence, CA simulation algorithms are to be implemented on multiprocessors. The preconceived opinion, that no much effort is required to obtain highly efficient coarse grained parallel CA algorithm, is not always true. In fact, a great variety of CA modifications coming into practical use need appropriate, sometimes sophisticated, methods of CA algorithms parallel implementation. Proceeding from the above a general approach to CA parallelization, based on domain decomposition correctness conditions, is formulated. Starting from the correctness conditions particular parallelization methods are developed for the main classes of CA simulation models: synchronous CA with multi-cell updating rules, asynchronous probabilistic CA, and CA compositions. Examples and experimental results are given for each case.

1 Introduction

A Cellular Automaton (CA) is nowadays an object of growing interest as a mathematical model for spatial dynamics simulation. In some fundamental works [1, 2] CA is expected to become a complement to partial differential equations due to its capability of simulating nonlinear and discontinuous processes. Particularly, CA may be helpful when there is no other way of representing a phenomenon to be simulated. By now a great variety of CA are known whose evolution simulate spatial dynamics of natural phenomena, which together with methods and tools for using them, are integrated under a unique concept referred to as *fine-grained parallelism.* The origin of fine-grained parallelism lays in classical CA theory. Classical CA have Boolean alphabet, deterministic one-cell updating transition functions, and synchronous mode of operation. They are capable to simulate diffusion, wave propagation, phase transitions, spatial self-organization [2, 3], etc. A more complicated class of CA called Lattice-Gas models [4] is used in hydrodynamics. In chemistry [5] and microelectronics [6] asynchronous probabilistic

⋆ Supported by 1)Presidium of Russian Academy of Sciences, Basic Research Program N 14.16 (2006), 2) Siberian Branch of Russian Academy of Sciences, Interdisciplinary Project 29 (2006)

CA (sometimes referred to as Monte-Carlo methods) are used for simulating real atoms and molecules moving and interacting.

It is clear that CA simulation of processes on micro-level requires very large CA (up to $10^{10} - 10^{12}$ cells) and many (up to 10^5) iterative steps to obtain a wanted information. Hence, parallel implementation on a multiprocessor system is essential, and methods for CA algorithms parallelization have been developed and studied. Intrinsic fine-grained parallelism and the results of a number of case studies has created an illusion that domain decomposition methods always provide highly efficient parallel programs requiring no much effort [7–9] . In fact, the above is true only for classical CA . As for its numerous modifications yet known and intensively emerging, the problem may be more complicated. For example, domain decomposition method is to be modified for CA with multi-cell updating rules [10, 12], the similar should be done when hybrid reaction-diffusion CA [11] is used. In case of asynchronous CA the operation mode is to be changed in order to achieve high efficiency of parallel implementation. At the same time, pursuing high efficiency it is possible to break the equality of the initial CA evolution and that of a decomposed CA. It follows therefrom that the problem is to be revised. In the paper an attempt to make such a revision is done based on *correctness conditions* which aim to guarantee the equality of initial CA evolution to that of its parallelized version. In fact, correctness conditions should provide conservation of transition rules of CA elementary automata involved in the interaction between processes. But, bearing in mind that CA mimics kinetics of real or virtual ("stylized ") particles, correctness conditions may also be regarded as conservation laws corresponding to the real process under simulation.

The paper aims to formulate correctness conditions for domain decomposition methods of CA parallelization, and to show how parallel algorithms should be developed for the most known types of CA simulation models. In the second section correctness conditions are formulated. The following three sections are devoted to domain decomposition algorithms of synchronous, asynchronous and hybrid CA decomposition algorithms which are illustrated by the examples from series of simulations of flow propagation through porous media of different type.

2 Correctness Conditions for CA Domain Decomposition

2.1 Formal Definitions

Cellular Automaton is defined by four terms, $\aleph = \langle A, M, \Theta, \rho \rangle$ [12], which have the following meaning: A is a state alphabet, M is a naming set, Θ is a local operator, ρ is a mode of functioning. The alphabet may be any set of numbers, symbols, vectors or matrices. The naming set used in CA-simulation comprises coordinate vectors of discrete space points (i, j, k), which are for short denoted by a single symbol m. Two particular sets A and M define a class of *cellular arrays* $A \times M$, whose each representative $\Omega = \{(a, m) : a \in A, m \in M\}$ is a set of pairs called *cells.* Each cell corresponds to an elementary automaton, named $m \in M$ in a state $a \in A$. On M naming functions $\phi(m) : M \to M$ are defined, whose values associate with a cell m a number of its neighboring cells, forming a *local configuration*

$$Q(m) = \{(u_0, m), ..., (u_k, \phi_k(m)), ..., (u_q, \phi_q(m))\}, \tag{1}$$

where $U_Q(m) = \{u_0, u_1, ..., u_q\}$ is a *state set* of $Q(m)$, and

$$T_Q(m) = \{m, \phi_1(m), ..., \phi_k(m), ...\phi_q(m)\} \tag{2}$$

is the *underlying template* for $Q(m)$.

Two local configurations

$$\begin{aligned} Q(m) &= \{(u_0, m), (u_1, \phi_1(m))..., (u_q, \phi_q(m))\}, \\ S(m) &= \{(v_0, \psi_0(m), (v_1, \psi_1(m), ...,)(v_s, \psi_s(m))\}, \end{aligned}$$

being composed into a substitution of the form

$$\Theta(m) : S(m) \to Q(m), \tag{3}$$

constitute a *local operator*, where $S(m)$ and $Q(m)$ are referred to *as a basic,* and a *next state* local configurations of Θ, respectively, m being called a *main cell* of Θ.

The next states $u_k \in U_Q$, $k = 0, ..., q$, are values of corresponding *transition function*

$$u_k = f_k(v_0, ..., v_s), \qquad k = 0, 1, ..., q. \tag{4}$$

Local operator Θ is applicable to a cell $m \in M$, if $T_S(m) \subseteq M$ and $v_k \in A$ for all $k = 0, ..., s$. Application of Θ to a cell $m \in M$ consists of two actions: 1) computing next states (4), and 2)updating cells of $Q(m)$ assigning the obtained values to their states.

A subset $M' \subseteq M$ referred to as a *main naming set* is defined, such that application of Θ to all $m \in M'$ comprises *an iteration* performing a global transition

$$\Theta(M') : \Omega(t) \to \Omega(t + 1), \tag{5}$$

The sequence

$$\Sigma(\Omega) = (\Omega, \Omega(1), ..., \Omega(t), \Omega(t + 1), ..., \Omega(\hat{t})), \tag{6}$$

obtained during iterative operation of the CA is called *the evolution*, t being the iteration number. CA evolution is the result of the simulation task, representing the process under simulation. If the process converges to a stable global state, then CA evolution has a termination, i.e. there exists such a $t = \hat{t}$, that $\Omega(\hat{t}) = \Omega(\hat{t} + 1) = \Omega(\hat{t} + 2) = ...$ If it is not so, then the evolution is infinite, exhibits oscillatory or chaotic behavior [2].

The mode $\rho \in \{\alpha, \beta, ..., \sigma\}$ of CA operation determines the order of cells to be chosen for local operator applications during the iteration. Synchronous (denoted as σ) and asynchronous (denoted as α) modes are the basic ones. Accordingly, a synchronous CA is denoted as $\aleph_\sigma$ and an asynchronous one - as $\aleph_\alpha$.

In synchronous CA cell-states of $\Omega(t)$ are updated only after all next states for all $m \in M$ are computed. Theoretically, it may be done in all cells simultaneously or at any order, which manifests the *cellular parallelism.* In fact, when a conventional sequential computer is used, such a cellular parallelism is imitated by delaying cell updating until all next states are obtained. So, really the cellular parallelism is a *virtual parallelism*, which cannot be for the benefit in conventional computers.

Asynchronous mode of operation suggests no simultaneous operation (neither real nor virtual). Intrinsic parallelism of $\aleph_\alpha$ is exhibited by the arbitrary order of cells to be chosen for application of $\Theta(m)$, the updating of cell states of $Q(m)$ being done immediately after $\Theta(m)$ is applied. So, each global transition $\Omega(t) \rightarrow \Omega(t+1)$ consists of $|M'|$ sequential acts of cell updating, being referred to as *global state transition sequences.* Due to random order of those acts the number of all possible transition sequences is qual to the number of transpositions in M, which is $\mu = |M|!$. The important property of asynchronous mode of operation is that the state values used by transition functions (4) may belong both to $\Omega(t)$ and to $\Omega(t+1)$. It is the reason why two CA - $\aleph_\sigma$ and $\aleph_\alpha$ with equal $\langle A, M, \Theta\rangle$ starting from the same Ω may have quite different evolutions. Although, some exotic "very good" CA are known, whose evolutions and attractors are invariant whatever mode of operation is used [12].

2.2 Correctness Conditions of CA Algorithms

A CA $\aleph_\varrho = \langle A, M, \Theta\rangle$ is considered to be a CA-algorithm, if its operation satisfies the following correctness conditions.

1. Noncontradictoriness. *Only one updating of a cell is allowed at one time step.*

Noncontradictoriness is a CA-version of *safeness* - a main property of parallel systems correct behavior. [12]. The sufficient condition of noncontradictoriness is as follows [12].

$$T_Q(m_k) \cap T_Q(m_l) = \emptyset \qquad \forall (m_k, m_l) \in M. \tag{7}$$

It guarantees the absence of conflicts, which are situations when two transition functions $f_g(m)$ and $f_h(\phi_l(m))$ are attempting to change the state of one and the same cell simultaneously. From (7) it follows, that for classical synchronous CA with $|Q(m)| = 1$ it is always satisfied. It is not so, if CA is a model of a process where some cells are to be changed simultaneously, i.e. $|Q(m)| > 1$. In this case a bit of cellular parallelism is to be sacrificed for noncontradictoriness by means of sequentializing the computation procedure as follows.

1) The main naming set M' is partitioned into b, $b \leq |T_Q|$, *stage-subsets,* $\{M'_0, ..., M'_b\}$, such that

$$M'_g \cap M'_h = \emptyset, \quad \forall (g,h) \in 1,...,b, \qquad \bigcup_{g=0}^{p} M'_g = M', \tag{8}$$

and for any M'_g, $g = 1, ..., b$, the condition (7) holds.

2) The iteration is divided into p stages. At each g-th stage Θ is applied synchronously to all cells $m \in M'_g$.

Such mode of operation is called a *multi-stage synchronous mode* with multi-cell updating and denoted as $\aleph_\beta$. As for asynchronous CA, they always satisfy noncontradictoriness conditions, because $\Theta(m)$ is applied to a single cell at each time-step.

Equality of cells. *At each iteration $\Theta(m)$ should be applied to all cells $m \in M'$, to any cell $m \in M'$ being applied only once.* Equality of cells is a CA-version of *liveness condition* for parallel processes [12]. It provides all cells to have equal rights to participate in the CA operation process. Synchronous classical CA satisfy this property by the definition of synchronicity. When multi-stage synchronous mode is used the equality of cells is provided by condition (8). In asynchronous CA cells equality is the consequence of binomial probability distribution law.

2.3 Correctness Conditions of CA Decomposition

Inherent cellular parallelism of CA models predetermines domain decomposition to be a basic principle for CA parallelization. In terms of CA this principle is read as follows. CA $\aleph_\varrho = \langle A, M, \Theta \rangle$ is represented by a composition of n ones, such that

1) $\aleph_\varrho^{(k)} = \langle A, M_k, \Theta \rangle$, $k = 1, ..., n$;

2) each domain M_k is a compact part of M;

3) domains do not intersect, i.e.

$$\bigcup_{k=1}^{n} M_k = M, \qquad M_k \cap M_l = \emptyset, \qquad \text{for all} \quad k, l \in \{1, ..., n\}; \tag{9}$$

4) all domains have equal cardinalities, $|M_1| = |M_2| = ... = |M_n|$.

5) It is convenient to assign cell names in the domains in such a way, that $(i, j) \in M_k$ is equal to $(i(modN_i), j(modN_j)) \in M$, where $N_i \times N_j$ is the size of Ω.

Since the result of CA simulation is its evolution, the main condition of coarse-grained parallelization is the equality of evolutions, i.e. *the evolution of n operating in parallel CA $\aleph_\varrho^{(k)} = \langle A, M_k, \Theta \rangle$, $k = 1, ..., n$; should be equal to that of a non-decomposed $\aleph_\varrho = \langle A, M, \Theta \rangle$* i.e.

$$\Sigma(\Omega) = \Sigma\left(\bigcup_{k=1}^{n} \Omega_k\right) \qquad \forall \Omega = \bigcup_{k=1}^{n} \Omega_k., \tag{10}$$

The condition being laid down, the problem is to organize the parallel operation in such a way that the condition is satisfied, i.e. make each domain to interact with the adjacent ones by exchanging data that are needed to be used in one of them for computing next-states in the other. To be more formal, let's denote as $(M_l,\, M_r)$ a pair of adjacent domains. Being applied to a cell $m_l \in M_l$, $\Theta(m_l)$ has to interact with the cells from $S(m_l)$, some of which, are allocated in M_r forming a *border area* of M_r denoted by Υ_r. The similar border area Υ_l comprises cells from M_l which are to be used by $\Theta(m_r)$.

$$\Upsilon_r = \bigcup_{m_l \in M_l} (T_S(m_l) \cap M_r), \quad \Upsilon_l = \bigcup_{m_r \in M_r} (T_S(m_r) \cap M_l). \tag{11}$$

To provide interactions between the domains two sets of cells $(\Upsilon_l', \Upsilon_r')$, whose cells are in one-to-one correspondence with those of (Υ_r, Υ_l), should be appended to the borders of $(M_l,\, M_r)$, respectively, forming the extended naming sets $\hat{M}_l$, and $\hat{M}_r$.

The procedure of data exchange consists of copying cell states of $m_l \in \Upsilon_l$ into its counterpart $\Upsilon_r' \subseteq \hat{M}_r$, and copying cell states of Υ_r into $\Upsilon_l' \subseteq \hat{M}_l$. Modes of the exchange procedure depend on the mode of $\aleph_\varrho$, but in all cases correctness properties is achieved by obeying the following *data exchange rules.*

1. At any iteration each cell state $m_l \in \Upsilon_l$ should be copied into the corresponding cell of its counterpart Υ_r'.

2. Between two acts of copying the state of $m_l \in \Upsilon_l$ into Υ_r', the cell m_l should be updated, and only once.

3. No cell in an appended area $m' \in \Upsilon_l'$ is allowed to be updated by $\Theta(m_l)$.

The first two rules provide the condition of cells equality in the adjacent border areas. The third ascertains that noncontradictoriness condition is not violated in border areas. T

From the above rules the method for allocating the a CA $\aleph_\rho = \langle A, M, \Theta \rangle$ to be run on n processors is as follows.

Step 1. Cut the cellular array into n compact equal parts with naming sets $\{M_k : k = 1, ..., n\}$ satisfying (9).

Step 2. Determine the border areas $\Upsilon_l \subseteq M_k$ and their counterparts $\Upsilon_r' \subseteq M_k$ according to (11) for all the borders of each domain.

Step 3. Develop the data exchange procedure according to the above four rules and to the mode of operation of $\aleph_\rho$.

It is the last step which constitutes the problem of parallelization, because the differential peculiarities of CA simulation models require special techniques to obey the above four data exchanging rules. For the most widely used CA-models the techniques are presented in the next section,

3 Parallelization of CA Algorithms

3.1 Synchronous CA Parallelization

The most simple for parallelization are synchronous CA-models with a single cell updating, $|Q(m)| = 1$. For them the procedure of data exchanging is trivial: at each iteration the border areas of adjacent domains are copied into their counterparts. It is easily seen that the procedure obeys all four data exchange rules. MPI tools, which are mainly used for performing data exchange, allow to make the transfer of data during the internal cells next states computation. Hence, the efficiency of parallel implementation is close upon 100%. It decreases only when the size of the domains is so small, that internal computation time does not exceed the time of data transferring [15]. The most known and well studied CA-models of this type belong to Lattice-Gas hydrodynamics [4]. The peculiarity of Lattice-Gas CA is in the fact that each iteration consists of two sequential stages (propagation and collision of particles). Since intercell communication occurs only at the propagation stage, the exchange of data may be done during the collision stage, which makes data exchange no time consuming procedure. Parallel implementation efficiency is thoroughly investigated in [14], where it is shown that degradation of the parallelization efficiency may occur due to small domain size and due to communication system problems.

As for synchronous CA-models with multi-cell updating ($|Q(m)| = q, q > 1$), the procedure of data exchange is more complicated, because noncontradictoriness conditions (7,8) require the following two statements to be taken into account.

- In each k-th domain the main naming set $M'_k \subseteq M_k$ is partitioned into p stage-subsets $\{M'_g : g = 1, ..., q\}$.

- Belonging to different stage-subsets M'_g, $g = 1, ..., q$ the main cells of $\Theta(m_l)$ $m_l \in M'_l$ occur at different distances from the domain border. Hence, border areas differ from stage to stage, $\Upsilon_l(g) \neq \Upsilon_l(h)$.

Based on the above statements data exchanging procedure is as follows.

1) In all domains the stage subarrays $M'_g \subset M_k$, $g = 1., , , .q$, are defined according to (9).

2) The iteration is divided into q stages. At each g-th stage Θ is applied to the cells of M'_g of all the domains.

3) For each g-th stage subarray, $g = 1, ..., q$, border areas $\Upsilon_l(g) \subseteq M_l$ and $\Upsilon_r(g) \subseteq M_r$ should be determined according to (11) for all borders pairs of adjacent domains (M_l, M_r), $l, r = 1., , , .n$.

4) Each domain M_k, $k = 1, ..., n$, should be appended by the counterparts $\Upsilon'_r(g)$ and $\Upsilon'_r(g)$ of the border areas $\Upsilon_l(g)$ and $\Upsilon_r(g)$, respectively, at all its borders.

5) At each g-th stage, $g = 1, ..., q$, the next cell states of border areas $\Upsilon_l(g) \subseteq M_l$ are copied into its counterpart $\Upsilon_r(g)' \subseteq M_r$ in all domains $M_k \in M$.

Example 1. A bright example of a CA with multi-cell updating is a Margolus' diffusion model $\aleph_\beta = \langle A, M, \Theta \rangle$, proposed in [3], and proved to be equivalent to Laplace PDE in [10]. $A = \{0, 1\}$, $M = \{(i, j) : i, j = 0, ..., N\}$. In M two subsets are defined: $M^{even} = \{(i, j) : i(mod_2) = 0, j(mod_2) = 0\}$ and $M^{odd} = \{(i, j) : i(mod_2) = 1, j(mod_2) = 1\}$. They induce on M two partitions by $2{\times}2$ blocks given by a template $T(i, j) = \{(i, j\}, (i, j{+}1), (i{+}1, j{+}1), (i{+}1, j)\}$. If $(i, j) \in M^{even}$ the template represents the even partition, otherwise it represents the odd one. The local operator is as follows

$$\begin{aligned} \Theta(i,j) :\ & \{(v_0, (i,j)), (v_1, (i, j+1)), (v_2, (i+1, j+1)), (v_3, (i+1, j))\} \\ & \rightarrow \{(u_0, (i,j)), (u_1, (i, j+1)), (u_2, (i+1, j+1)), (u_3, (i+1, j))\}, \end{aligned} \tag{12}$$

$$\text{where } u_k = \begin{cases} v_{k-1}(mod_4), & \text{with probability} \quad \pi \\ v_{k+1}(mod_4), & \text{with probability} \quad 1-\pi \end{cases}, \qquad k = 0,1,2,3.$$

The mode of operation is two-stage synchronous, i.e. at even t $\Theta(i,j)$ is applied to M^{even}, at odd t – to M^{odd}. The model is used for simulating flow propagation of water through a porous substance under the influence of isotropic diffusion. (Fig. 1).

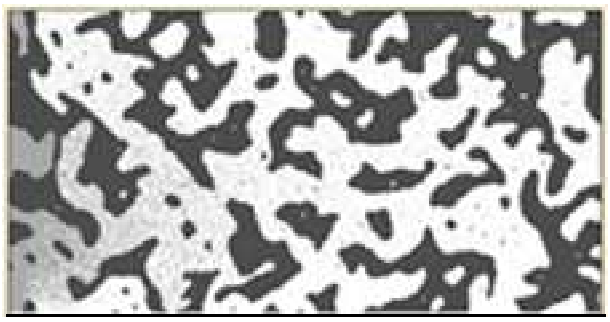

Fig. 1. A snapshot ($t = 4 \cdot 10^5$) of the simulation process or flow propagation in porous medium under isotropic diffusion. A fragment 300 × 600 cells is shown. Black cells correspond to solid walls, grey pixels - to fluid, white - to empty space.

Proceeding from the given physical parameters of the sample under simulation the cellular naming set is chosen as $M = \{(i,j : i = 0,...,8N, j = 0,...,N\}$, with $N = 999$. According to the above method the parallel algorithm is as follows.

1. The cellular array is decomposed into $n = 16$ domains with naming sets $M_k = \{(i,j) : i = 0,...,499, j = 0,...,999\}$, $k = 1,...,n$, and $N(mod_2) = 0$.

2. On each domain M_k, $k = 1,...,16$, two subsets of names $M^{even} \subseteq M_k$ and $M^{odd} \subseteq M_k$ are defined.

3. Border areas and their counterparts are determined for the even and the odd stages as follows.

Even stage. Since the number of cells in any domain is even, then according to (11) for any pair of adjacent domains $M_l^{even} \subseteq M_l, M_r^{even} \subseteq M_r$ $T_S(i,j)_l^{even} \cap M_r^{even} = \emptyset$ for all $((i,j)_l(even)) \in M_l$, which yields in

$$\Upsilon_l = \Upsilon_r = \Upsilon_l' = \Upsilon_r' = \emptyset$$

Odd stage. The underlying template $T_S(N-1,j) = \{(N-1,j)(N-1,j+1)(N,j)(N+1,j)\}$ of the border cells of M_l indicates that cells included in two last terms – (N,j) and $(N+1,j)$ – are allocated in M_r, being named there as $\{0,j),(0,j+1)\}$, j=0,...,N-1. Substituting it into (11) yields for j=0,...,N-1,

$$\Upsilon_r = \{(0,j)\}, \quad \Upsilon_l = \{(N-1,j)\}, \quad \Upsilon_l' = \{(N,j)\}, \quad \Upsilon_l' = \{(-1,j)\},$$

and Υ_l' and Υ_r' are appended adjoining the borders of M_l and M_r, respectively. The similar is true for all adjacent borders.

4. After the even stage is completed no exchanges are done because the border areas for even stage are empty.

5. After the odd stage is completed data exchange is performed between all adjacent pairs of the domains: cell states of Υ_l are copied to the corresponding cells of Υ_r' and cell states of Υ_r are copied to the corresponding cells of Υ_l'.

The algorithm has been programmed and implemented in 16 processors of the cluster MVS-1000/128 in Siberian Supercomputer Center. Implementation of the algorithm in 16 processors showed the run time 1.012 times greater than that of a CA with the array size of one domain.

3.2 Asynchronous CA Parallelization

As distinct from the CA, whose parallel implementation is extremely efficient, the asynchronous case exhibits a problem for parallel simulation. The reason is in the impossibility of forming packages for data exchange. Each state $m_l \in \Upsilon_l$ is to be copied to Υ_r' of the adjacent domain just after the cell is updated. No delay for even a single time-step τ is allowed, because at the same time-step the cell (u, m_l) may be updated by the application of Θ to its neighbor, which violates the noncontradictoriness condition. It is evident that transferring data to adjacent processor after each updating of a border cell results in a very low efficiency of parallel implementation, because of transfer latency time, which is usually some orders larger than the transmission of a data bit. Moreover, in those random intercommunications one should avoid the deadlocks, which is an additional task. So, the above direct data exchange method should be rejected. The advantages of synchronous CA parallel implementation inspires the search of a transformation of the given asynchronous CA into a synchronous one having the same evolution. Unfortunately, there is no transformations which provide equality of evolutions in general case. Thus, the attempt is made to find a multi-stage synchronous CA, whose evolution approximates that of a given asynchronous $\aleph_\alpha$. It has been used for particular cases in [16, **?**], and considered in detail in [15]. The term "approximation" is used in the following sense. Some order is imposed to the random choice of cells to be updated, which brings no distortion in the evolution progress, but only restricts the ensemble of all possible transition sequences to the next global state. The algorithm for constructing $\aleph_\beta =$ which approximats the given $\aleph_\alpha = \langle A, M, \Theta \rangle$ is as follows.

1. Parameters A, M, and Θ are the same than those of $\aleph_\alpha$,where $\Theta : S(m) \to Q(m)$ with $|T_Q(m)| = q$.

2. A template T_B is defined, such that

$$T_B(m) \supseteq T_Q(m). \tag{13}$$

Naturally, $T_B(m)$ should be chosen of minimum cardinality, because it results in a less amount of stages.

3. On the main naming set $M' \in M$ the subsets $\{M'_0, ..., M'_b\}$, $b = |T_B(m)|$ are defined satisfying the condition (8) for multi-stage CA. Moreover, for any M'_g, $g = 1, ..., b$, the noncontradictoriness condition should be met, i.e.

$$T_B(m_k) \cap T_B(m_l) = \emptyset \qquad \forall (m_g, m_h) \in M_g, \qquad \bigcup_{m_g \in M_g} T_B(m_g) = M. \tag{14}$$

4. Each iteration is divided into b stages. At each gth stage Θ is applied synchronously to all cells $m \in M'_g$, the subsets M'_g being chosen in any order.

In [15] it is proved that $\aleph_\beta$ obtained by the above algorithm is the restriction of $\aleph_\alpha$ in the sense that the set of its evolutions is included in the ensemble of all possible evolutions of $\aleph_\beta$, being far less in cardinality. By the above transformation he parallelization method of the $\aleph_\alpha$ is reduced to that of multi-stage CA parallelization.

Example 2.When anisotropy imposed by the pore walls properties and an additional pressure are to be taken into account, probabilistic asynchronous diffusion CA called a "naive diffusion" is used, $\aleph_\alpha = \langle A, M, \Theta \rangle$, $A = \{0, 1\}$, $M = \{(i, j)\}$, The application of Θ to a cell $(i, j) \in M$ makes the cell (i, j) to exchange states with one of its nearest neighbors $\phi_k(i, j)$ chosen with probability $p = p_k$, $k = 1, 2, 3, 4$.

$$\begin{array}{ll} \Theta(i,j) : & \{(v_0, (i,j)), (v_1, (i-1,j)), (v_2, (i,j+1)), (v_3, (i+1,j)), (v_4, (i,j_1))\} \\ \rightarrow & \{(u_0, (i,j)), (u_1, (i-1,j)), (u_2, (i,j+1)), (u_3, (i+1,j)), (u_4, (i,j_1))\}, \end{array}$$

where

$$(u_0 = v_k)\&(v_k = u_0) \quad \text{if} \quad \begin{cases} (v_0 = 1\&v_k = 0) & \text{with} \;\; p = p_k, \\ (v_0 = 0\&v_k = 1) & \text{with} \;\; p = 1 - p_k. \end{cases}$$

The probabilities are determined by physical properties of the medium. Particularly, in case of hydrophobic pores and presence of a convective flow in the direction of the jth axis they are as follows.

$$p_1 = p_3 = 0.25p_d, \quad p_2 = 0.3p_d, \quad p_4 = 0.2p_d \quad p_d = \sin \frac{\pi}{20} d,$$

where d is the distance between the cell (i, j) and the nearest wall. The local operator is applied to all cells of the array, i.e. $M' = M$. Parallel application of the model has been tested in simulation of flow propagation through the sample of porous substance having the same dimensions than those of Example 1. The parallel algorithm consists of two phases: 1)constructing the multistage approximation $\aleph_\beta$, and 2) determining the parameters of the data exchange procedure.

Phase 1.

1. The template $T_B = \{(i+k, j+l) : k, l = -1, 0, 1\}$, satisfying (12) is defined with $s = |S(m)| = 9$.

2. Stage-subsets M_g, g=0,1,...,8, are formed according to (8) as follows:

$$(i, j) \in M_g \quad \text{if} \quad g = 3i(mod_3) + j(mod_3), \tag{15}$$

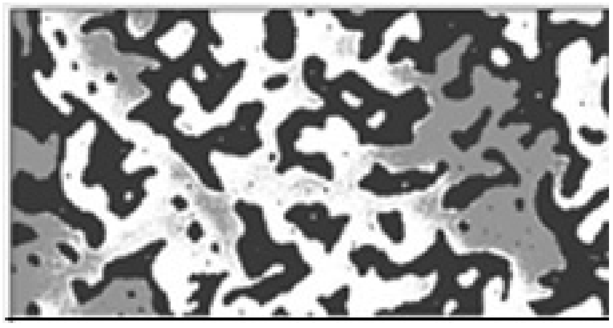

Fig. 2. A snapshot ($t = 70 \cdot 10^3$) of the simulation process or flow propagation in hydrophobic porous medium under anisotropic diffusion. A fragment 300×600 cells is shown. Black cells correspond to solid walls, grey pixels - to fluid, white pixels - to empty space.

Phase 2

1. The array is decomposed in 16 domains of 501×1002 cells, $N = 501$ is chosen being a multiple to $|S(m)| = 3 \times 3$, which allows to distribute the cells of M among the stage-subsets according to the rule (15).

2. Border areas and their counterparts are computed following (11) as follows.

$$\begin{array}{llll}
\text{if } g(mod_3) = 0: & \Upsilon_l = \Upsilon_r' = \emptyset, & \Upsilon_r = \{(i, N-2), (i, N-1), & \\
 & & \Upsilon_l' = \{(i,-1), (i,-2)\}, & \\
\text{if } g(mod_3) = 1: & \Upsilon_l = \Upsilon_r' = \emptyset, & \Upsilon_r = \Upsilon_l' = \emptyset, & \\
\text{if } g(mod_3) = 2: & \Upsilon_r = \Upsilon_l' = \emptyset, & \Upsilon_l = \{(i,0), (i,1)\}, & \Upsilon_r' = \{(i,N), (i,N+1),\} \\
\text{for} \quad i = 0, ..., N-1. & & &
\end{array} \tag{16}$$

The algorithm has been programmed and implemented in 16 processors of the cluster MVS-1000/128 in Siberian Supercomputer Center. The running time in 16 processors is 1.056 times greater than that for running the same amount of iterations in a single domain in a single processor.

3.3 Parallelization of Composed CA

Real life simulation tasks require several simple CA-models to operate in common for being adequate to a phenomenon under study. A number of methods are known [18] for composing some simple CA to obtain a CA-model of a complicated phenomenon. Two basic methods are the most used: a sequential composition called *superposition*, and a *parallel composition*, which are worth to be considered concerning coarse grained parallelization.

When *superposition* is used the process under simulation is composed of n component CA $\aleph_\rho^{(k)} = \langle A, M, \Theta^{(k)} \rangle$, $k = 1, ..., n$, which have identical alphabets and naming sets, but may differ in local operators and modes of operation. The composed CA $\aleph_\rho = \langle A, M, \Phi \rangle$, $\Phi = \{\Theta^{(1)}, ..., \Theta^k, ..., \Theta^n\}$, operates as follows.

Each iteration $\Omega(t) \xrightarrow{\Phi} \Omega(t+1)$ consists of n sequential transitions $\Omega^{(k)}(t) \xrightarrow{\Theta^{(k)}} \Omega^{(k)}(t)$, $k = 1, ..., n$, each kth transition being an iteration of $\aleph_\rho^{(k)}$. It is worth to notice, that any component CA performs its transition operating in its own mode. Moreover, a component CA may be itself a composed CA, in what case the composition exhibits an hierarchial construction. The method of parallelization of a global superposition reduces to construction of the iteration of $\aleph_\rho$ as a sequence of n iterations of the parallel algorithms of $\aleph_\rho^{(k)}$, developing the data exchange procedures according to the rules, corresponding to their modes of operation.

More complicated is coarse grained parallelization of CA which is a *parallel composition*, when two[1] CA operate each on its own cellular array, using cell states of the other as variables in its transition functions. Let two component CA be $\aleph_\rho^{(1)} = \langle A^{(1)}, M^{(1)}, \Theta^{(1)} \rangle$, and $\aleph_\rho^{(2)} = \langle A^{(2)}, M^{(2)}, \Theta^{(2)} \rangle$. They should have identical modes of operation, identical naming sets, but may have different alphabet and different local operators.

$$\Theta^{(1)}((i,j)^{(1)}) : S^{(1)}((i,j)^{(1)}) \to Q^{(1)}((i,j)^{(1)}),$$
$$\Theta^{(2)}((i,j)^{(2)}) : S^{(2)}((i,j)^{(2)}) \to Q^{(2)}((i,j)^{(2)}).$$

The basic template

$$T_{S^{(1)}}(m) = \{\phi_0(m), ..., \phi_l(m), \phi_{(l+1)}(m), ..., \phi_s(m)\}, \quad m \in M^{(1)}$$

has the first l naming functions defined in $M^{(1)}$, and the last $s-l$ ones – defined in $M^{(2)}$. Similarly,

$$T_{S^{(2)}}(m) = \{\phi_0(m), ..., \phi_h(m), \phi_{h+1}(m), ..., \phi_g(m)\}, \quad m \in M^{(2)}$$

has the first h naming functions defined in $M^{(2)}$, and the last $g-h$ ones – defined in $M^{(1)}$. Each t-th iteration of a composed CA comprises next state computation in all cells of both CA.

Parallelization method for a composed CA should follow all the rules given in Subsection 2.3, being slightly modified as follows.

Step 1. Cellular arrays of both CA are cut into n compact equal parts $M_k = M_k^{(1)} \cup M_k^{(2)}$.

Step 2. Border areas are determined according to (11) for all pairs of adjacent domains $M_l^{(1)}, M_r^{(1)}$ and $M_l^{(2)}, M_r^{(2)}$.

$$\begin{aligned} \Upsilon_r^{(11)} &= \bigcup\nolimits_{m_l \in M_l^{(1)}} \left(T_S^{(1)}(m_l) \cap M_r^{(1)}\right), \ \Upsilon_r^{(12)} = \bigcup\nolimits_{m_l \in M_l^{(1)}} \left(T_S^{(1)}(m_l) \cap M_r^{(2)}\right), \\ \Upsilon_r^{(22)} &= \bigcup\nolimits_{m_l \in M_l^{(2)}} \left(T_S^{(2)}(m_l) \cap M_r^{(2)}\right), \ \Upsilon_r^{(21)} = \bigcup\nolimits_{m_l \in M_l^{(2)}} \left(T_S^{(2)}(m_l) \cap M_r^{(1)}\right), \end{aligned} \tag{17}$$

[1] The amount of component CA is confined to 2 because there is no experience of testing parallel composition of more than 2 CA, though there is no principal objection for the method to be extended to any numbers of component CA

Their counterparts $\Upsilon_l'^{(11)} \cup \Upsilon_l'^{(12)}$ are to be appended to $M_l^{(1)}$, and $\Upsilon_l'^{(22)} \cup \Upsilon_l'^{(21)}$ - to $M^{(2)}$. The similar should be done to all other borders of both domains.

Step 3. Data exchange procedure consists of copying cell states from all border areas of each component array to their counterparts in the adjacent domains.

The above global parallel composition is used for simulation reaction-diffusion phenomena, where diffusion may be modeled by a Boolean CA $\aleph_\rho^{(1)})$, and reaction - by a CA with real alphabet [11] At each iteration transition functions (4) of $\aleph_\rho^{(1)}$ have to transform real variables from $\Omega^{(2)}$ into Boolean form in order to compute Boolean function. On its turn $\aleph_\rho^{(2)}$ has to transform Boolean cell states of $\Omega^{(1)}$ into reals for computing its transition functions. The latter transformation includes averaging the Boolean states over the given *averaging area* which plays the role of basic local configuration $S^{(2)}(m)$, which is allocated in $\Omega^{(2)}$.

Example 3. Simulation of flow propagation through porous medium, where the fluid is exposed to a chemical reaction (oxidation), is modeled by a parallel composition of $\aleph_\beta^{(1)} = \langle A^{(1)}, M^{(1)}, \Theta^{(1)} \rangle$, simulating Boolean isotropic diffusion , and $\aleph_\sigma^{(2)} = \langle A^{(2)}, M^{(2)}, \Theta^{(2)} \rangle$ simulating reaction in reals. The diffusion CA $\aleph_\beta^{(1)} =$ in its turn is the superposition of $\aleph_{trans}^{(1)} = \langle \{0,1\}, M^{(1)}, \Theta_{trans} \rangle$, which performs transformation of real array $\Omega^{(2)}$ into a Boolean form, and $\aleph_{diff} \langle \{0, 1\, M^{(1)}, \Theta_{diff} \rangle$ simulating diffusion of Example 1, Θ_{diff} being equal to (12).

$$\begin{array}{l} \Theta_{trans} : S_1^{(1)}(m^{(1)}) \to Q_1^{(1)}(m^{(1)}), \text{ where} S_1^{(1)}(m^{(1)}) = \{(v, m^{(1)}, (u, m^{(2)})\}, \\ Q_1^{(1)}(m^{(1)}) = \{(f_1(u), m^{(2)})\}, \qquad f_1(u) = 1 \ \text{ with } \ \pi = u. \end{array}$$

The local operator $\Theta^{(2)}$ is applied to the cells of $\Omega^{(2)}$ with states in $A^{(2)} = [0, 1]$, where

$$\begin{array}{ll} S^{(2)}(m)^{(2)} & = \{(u, m^{(2)}), (v_0, m^{(1)}), (v_1, \phi_1(m^{(1)}), ..., (v_s, \phi_s(mm^{(1)})\}, \\ Q^{(2)}(m) & = \{(u\prime, m^{(2)}\}, \end{array}$$

where s is the averaging area size $5 \times 5 - 1$, $s = 24$ and

$$u\prime = 0.2w(1 - w), \quad w = \frac{1}{s} \sum_{l}^{s} v_k.$$

Parallel application of the model has been tested on the flow propagation through the sample of porous medium having the same size than in Example 1. An iteration of the parallel algorithm is as follows.

Step 1. Both arrays are decomposed into 16 domains 500×1000 cells $\{M_k\}$, $k = 1, ..., 16$.

Step 2. Border areas and their counterparts are determined according to (11).

$$\Upsilon_l^{(1)}(\Theta_{trans}) = \Upsilon_r^{(1)}(\Theta_{diff}) = \emptyset,$$

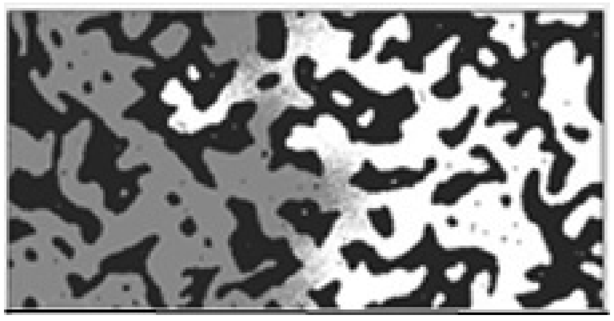

Fig. 3. A snapshot($t = 120 \cdot 10^3$) of the simulation process of flow propagation in porous medium under isotropic diffusion and oxidation. A fragment 300×600 cells is shown. Black cells correspond to solid walls, grey pixels - to fluid, white pixels - to empty space.

$\Upsilon_l^{(1)}(\Theta_{diff})$ and $\Upsilon_r^{(1)}(\Theta_{diff})$ are equal to those from Example 1 (step 3).

$$\begin{aligned} \Upsilon_r^{(1)}(\Theta^{(2)}) &= \{(i,j) : i = 0, ...5\} & \Upsilon_l^{(1)}(\Theta^{(2)}) &= \{(i,j) : i = N-1, ..., N-6\}, \\ \Upsilon_r^{(1)\prime}(\Theta^{(2)}) &= \{(i,j) : i = -1, ... -5\}, & \Upsilon_l^{(1)\prime}(\Theta^{(2)}) &= \{(i,j) : i = N, ..., N+5\}, \\ & & j &= 0, .., N-1. \end{aligned}$$

Step 3. In all domains M_k next states of $\Omega_k^{(1)}$ and $\Omega_k^{(2)}$ are computed and data exchange is performed between all pairs $(M_l^{(1)}, M_r^{(1)})$ of the adjacent domains.

Implementation of the algorithm in 16 processors showed the run time to be 1.021 times greater than that of the same simulation with the size 16 times less.

4 Conclusion

A general approach to domain decomposition methods for coarse-grained parallellization of CA algorithms is proposed. The approach is based on the fundamental principles of parallel processes correct behavior, which are formulated in the form of conditions to be met when organizing data exchange between domains. It is shown that the intrinsic cellular parallellism of CA-models does not garantee simple and correct coarse-grained parallelization methods, which differ esentially for different modes of CA operation. For asynchronous mode of operation high parallelization efficiency may be acheived by transformation CA in a multi-stage synchronous one. At any case parallelization efficiency is close to 0.9-1.

References

1. Toffolli, T.: Cellular Automata as an Alternative to (rather than Approximation of) Differential Equations in Modeling Physics. Physica D, Vol.10 (1984) 117-127.

2. Wolfram, S.: A new kind of science. Wolfram Media Inc., Champaign, Ill., USA (2002)
3. Toffolli T., Margolus N.: Cellular Automata Machines. USA, MIT Press (1987)
4. Rothman, B.H., Zaleski, S.: Lattice-Gas Cellular Automata. Simple Models of Complex Hydrodynamics. London: Cambridge Univ. Press. (1997)
5. Latkin, E.I., Elokhin, V.I.,Gorodetskii, V.V.: Spiral concentration waves in the Monte-Carlo model of CO oxidation over Pd(110) caused by synchronization via CO_{ads} diffusion between separate parts of catalytic surface. Chemical Engineering Journal, Vol.91 (2003) 123-131
6. Neizvestny,I.G., Shwartz, N.L., Yanovitskaya,Z.Sh., and Zverev A.V.: 3D-model of epitaxial growth on porous {111} and {100} Si surfaces. Computer Physics Communications, Vol.147 (2002) pp.272-275
7. Sipper, M.: Evolution of Parallel Cellular Machines: The Cellular Programming Approach. Springer-Verlag, Berlin Heidelberg, (1997)
8. Bandini, S., Erbacci, G., Mauri, G.: Implementing Cellular Automata Based Models on Parallel Architectures: The CAPP Project. Malyshkin, V.(ed): Lecture Notes in Computer Science, Vol.1662 (1999) 167-179.
9. Carotenuto, L., Mele,F., Furnari, M.M., and Napolitano, R.: Pecans: A parallel environment for cellular automata modeling. Complex Systems, Vol.10 (1996) 23-41
10. Malinetski, G.G., Stepantsov, M.E.: Modeling Diuffusive Processes by Cellular Automata with Margolus Neighborhood. Zhurnal Vychislitelnoy Matematiki i Matematicheskoy Phiziki, Vol.36, N 6 (1998) 1017-1021 (in Russian).
11. Bandman, O.: Simulation Spatial Dynamics by Probabilistic Cellular Automata. Chopard,B.(ed.). Lecture Notes in Computer Science, Vol. 2493. Berlin, Springer (2002) 10-19.
12. Achasova, S., Bandman, O., Markova, V., Piskunov, S.: Parallel Substitution Algorithm. Theory and Application. - Singapore, World Scientific (1994)
13. Park, J.K., Steiglitz, K., Thurston, W.P.: Soliton-like behavior in automata. Physica D, Vol.19 (1986) 423-432
14. Medvedev, Yu. G.: Experimental study of Computational characteristic of parallel implementation of 3D cellular Automata model of viscous flow. In: Proceedings of Scientific Confernce "Parallel Programming Technology", Moscow Univ. Press(2006) 79-82.
15. Bandman, O.: Parallel Implementation of Asynchronous Cellular Automata Algorithm. Yacoubi S.El, Chopard B.,Bandini, S.(eds): ACRI-2006, Lecture Notes in Computer Science, Vol.4173. Berlin, Springer (2006) 41-47.
16. Nedea, S.V., Lukkien, J.J., Jansen, A.P.J., and Hilbers, P.A.J.: Methods for parallel simulation of surface reaction. Werner, B.(ed.) 4th Int. Workshop on Parallel and Distributrd Scientific and Engineering Computing with Applications: IEEE Comp. Society, Nice, France (2003) 7-16
17. Chen N., Glazier J.A., Alber, M.S. A Parallel Implementation of the Cellular Potts Model for Simulation of Cell-Based Morphogenesis Yacoubi S.El, Chopard B.,Bandini, S.(eds): ACRI-2006, Lecture Notes in Computer Science, Vol.4173. Berlin, Springer (2006) 58-67.
18. Bandman, O. Composing Fine-graned Parallel Algorithms for Spatial Dynamics Simulation Malyshkin, V.(ed.): PaCT-2005, Lecture Notes in Computer Science, Vol. 3606. Berlin, Springer (2005) 99-113.