Bull. Nov. Comp. Center, Comp. Science, 46 (2022), 1–11

Development and parallel implementation of cellular automata phase separation models with an integer state alphabet

K.A. Glinskiy

Abstract. The paper is devoted to the development of two new cellular automata models of phase separation with synchronous and asynchronous modes. Cell transition functions with integer alphabet of states for these models are formulated. Their sequential and parallel program implementations based on the library of cellular automata topologies have been proposed. The simulation results of the new models were compared with each other and with the model with binary alphabet of cell states. For the new models, a qualitative correspondence with the physical view for liquids with high viscosity was obtained.

Introduction

Cellular automata simulation started to develop relatively recently. Many processes have not been described by means of cellular automata models yet. Therefore, the construction of new models is an actual task.

The work is devoted to the creation of two cellular automata with an integer alphabet of ell states taken as the models of phase separation.

The simulated physical process proceeds as follows. If you pour a well-mixed mixture of two immiscible liquids with different surface tension coefficients into a flat vessel, the separation of phases will begin: the first (lighter) liquid floats and it tends to gather into large spots on the surface of the second (heavier) liquid.

Traditionally in colloid chemistry this process is described using Newton's, Poiseuil's and Einstein's laws [1]. There are a number of cellular automata models of this process. For example, the model of phase separation with binary alphabet is described in [2]. This model shows the distribution of liquids on the surface quite realistically, but the masses of liquids are not conserved in this type of simulation.

The objective of this work is to develop both of synchronous and asynchronous cellular automata models of phase separation with saving the mass of liquids.

In order to do this, it is necessary to:

1. Formulate transition functions for each model, using the integer alphabet of cell states.

2. Perform software implementation of the models using the library of cellular automata topologies [3].
3. Perform computational experiments. Compare the results derived from the new models with each other and with the existing model of phase separation with binary alphabet.
4. Reveal the limitations of the developed models and formulate the further direction of elaboration.

The paper presents two models, the first one is based on a synchronous cellular automata, while the second model is based on an asynchronous one (these models are described in Section 1). The software implementation of these models is considered in Section 2. The results of computational experiments are presented in Section 3.

1. Model description

The models presented in this paper are an improvement of the existing cellular automata model for phase separation with a binary alphabet. The models have an integer alphabet of cell states $X = \{0, 1, \ldots, X_{\max}\}$. State $x \in X$ is interpreted as the amount (the layer thickness) of the first liquid on the surface of the other one, expressed in relative model units being proportional to the physical amount of the liquid. The amount of the liquid K in a cell corresponds to the thickness of the layer at which the equilibrium of gravity and surface tension forces is achieved, and is chosen arbitrarily from the range $1 \leq K < X_{\max}$.

The cellular array lies in a two-dimensional plane, and the cells are square in shape. The transition function of this model uses a first-order Moore neighborhood, with each cell interacting with eight nearest neighbors.

1.1. Transition function of synchronous model. The function of cell transitions in the synchronous model is given by the following rules:

1. Algorithm 1 presents pseudocode of this rule:

   If the state of the cell $x > K$, it is left with the amount of the first liquid $x' = K$, and the value $x - K$ is distributed equally among the selected neighbors. Neighbors are chosen from those whose state $x_i < K$, $i = 1, \ldots, 8$. The value x_i of each selected neighbor must not exceed x_j of any unselected neighbor. The number of selected neighbors N_{sel} should be as large as possible, provided that the value $\lfloor (x - K + \sum x_i)/N_{\text{sel}} \rfloor$ does not exceed x_j of each unselected neighbor. Then the remainder of the division $(x - K + \sum x_i) \bmod (N_{\text{sel}})$ is given to a random neighbor.

2. Algorithm 2 presents pseudocode of this rule:

   If the cell state $x = K$, and it has one or two neighbors with states $x_i \geq K$, $i = 1, \ldots, 8$, then this cell chooses a neighbor whose state x_j is maximal among all neighbors with states $x_j < K$, $j = 1, \ldots, 8$, and gives him a value $K - x_j$, so that this neighbor's state becomes $x'_j = K$, and leaves itself a value $x' = x_j$.

3. Algorithm 3 presents pseudocode of this rule:

   If the cell state $0 < x < K$, and it has no neighbors with state $x_i \geq K$, $i = 1, \ldots, 8$, then it distributes the value x among its neighbors, first to the neighbor with the maximum x_i, and then to the neighbors standing next to it. Each of the neighbors is given no more than $K - x_i$. Then the cell will go to the state $x' = 0$.

4. In the other cases the states of cells remain unchanged: $x' = x$, $x'_i = x_i$, $i = 1, \ldots, 8$.

Algorithm 1. Cell transition function. Rule 1 ($x > K$).

```
procedure phaseSep1(x, K, neighbors[1..8])
  sum ← x − K
  Nneighbors ← 0
  neighborsID[1..8] ← 0
  for i = 1 to 8 do
    if K > neighbors[i] then
      sum ← sum + neighbors[i]
      Nneighbors ← Nneighbors + 1
      neighborsID[Nneighbors] ← i
  flag ← 0
  while flag = 0 do
    if Nneighbors = 0 then
      break
    flag ← 1
    add ← sum/Nneighbors
    remainder ← sum % Nneighbors
    for i = 1 to Nneighbors do
      if neighbors[neighborsID[i]] > add then
        flag ← 0
        sum ← sum − neighbors[neighborsID[i]]
        Nneighbors ← Nneighbors − 1
        for j = i to Nneighbors do
          neighborsID[j] ← neighborsID[j + 1]
        break
  rand ← randomFrom(1, Nneighbors)
```

```
  for i = 1 to Nneighbors do
    if rand = i then
      value ← add + remainder − neighbors[neighborsID[i]]
    else
      value ← add − neighbors[neighborsID[i]]
    addValueToNeighbors(value, neighborsID[i])
  x ← K
```

Algorithm 2. Cell transition function. Rule 2 ($x = K$).

```
procedure phaseSep2(x, K, neighbors[1..8])
  max ← 0
  Nneighbors ← 0
  for i = 1 to 8 do
    if neighbors[i] < K then
      if neighbors[i] > max then
        max ← neighbors[i]
        IDmax ← i
      else
        Nneighbors ← Nneighbors + 1
        if Nneighbors > 3 then
          return
  addValueToNeighbors(K − max, IDmax)
  x ← max
```

Algorithm 3. Cell transition function. Rule 3 ($0 < x < K$).

```
procedure phaseSep3(x, K, neighbors[1..8])
  max ← 0
  Nneighbors ← 0
  for i = 1 to 8 do
    if neighbors[i] ≥ K then
      return
    if neighbors[i] > 0 then
      if neighbors[i] > max then
        max ← neighbors[i]
        IDmax ← i
      Nneighbors ← Nneighbors + 1
  if Nneighbors = 0 then
    return
  sum ← x
  while sum > 0 do
    part ← K − neighbors[IDmax]
    if sum > part then
```

```
      addValueToNeighbors(part, IDmax)
      sum ← sum − part
      IDmax ← IDmax % 8
      IDmax ← IDmax + 1
    else
      addValueToNeighbors(sum, IDmax)
      sum ← 0
x ← 0
```

1.2. Transition function of asynchronous model. The function of cell transitions in the asynchronous model is given by the following rules:

1. If $\sum_{i=0}^{8} x_i \geq 9K$, it is distributed evenly between the cell and its neighbors.
2. Otherwise, it is distributed among the cells in descending order of x_i by K, except perhaps the last one, which gets less liquid than K.

2. Software implementation

The models are implemented in C [4] as three modules — preprocessor, simulator, and postprocessor — using the library of cellular automata topologies [3].

2.1. Preprocessor. This module sets the size of the array, the proportion of mixture components and the K level when the equilibrium of gravity and surface tension is reached. The preprocessor creates a cellular array, fills some of the cells with mixture components in an amount from 1 to $X_{\max}$ equally and writes the cellular array to a file. Parameters such as array size, number of iterations, and cell size are also written to the same file.

The preprocessor uses the following library functions:

- CAT_InitPreprocessor

  This function takes as input the specified topology, the size of the data area occupied by one cell, a size of common data area available to each cell in the array independently of their neighborhood, and the width and length of the cellular array.

  The task of the function is to initialize the cellular array and generate a binary file header in RAM.

- CAT_PutCell

  This function takes as input a pointer to the structure with the state of a cell and the cell indices in the array.

  The function passes the specified values to the cell of the array by the given index.

- CAT_FinalizePreprocessor

  This function takes as input the file name where the cell array information will be written.

  It writes the generated cellular array into the binary file and frees the memory used to store the cell array.

2.2. Simulator. The simulator reads the cellular array state from the file, performs a specified number of iterations on it, and writes the resulting cellular array state to the file.

In the synchronous mode, the cell transitions function is automatically applied to all cells of the array at each iteration by the library. This ensures that each cell of the array interacts with all eight neighbors. In the asynchronous mode, the cells are selected randomly.

Periodic boundary conditions are applied to the array, that is, the leftmost neighbors of the cells of the leftmost column are their corresponding cells of the rightmost column and vice versa. The same principle applies to the upper and lowermost rows.

During the work on the software implementation for these models, the library of cellular automata topologies was modified to speed up the program execution. To speed up the computation of new cell values, for the synchronous mode, a parallel implementation of its cell array traversal module in a system with shared memory was performed using OpenMP.

The simulator uses the following library functions:

- CAT_InitSimulator

  This function takes as input a binary file containing information about the cell array and the simulation mode (synchronous or asynchronous).

  The function reads the contents of the cellular array from the file and puts the read array into memory.

- CAT_SetNumThreads

  This function takes as input the number of threads.

  Sets the number of threads used to run the transition function in parallel.

- CAT_Iterate

  This function takes as input the address of the cell transitions function.

  It applies the transition function to proper cells of the array according to the simulation mode. It needs to be called repeatedly, according to the number of iterations.

- CAT_FinalizeSimulator

  This function takes as input the name of the file in which the updated cell array information will be written.

  The function writes the updated cellular array to the binary file and frees the memory used to store the cellular array.

2.3. Postprocessor. In this module, the state of the cellular array is read from the file, the results are processed and the calculated values (the number of mixture components in each cell) are output in a user-friendly format.

To visualize the cellular array, a function has been implemented that creates an image with a grayscale color palette as a pgm file [5] based on the cell states. The lighter the cell, the higher the content for the first mixture component, and, vise versa, the darker color means a lower content for this component. Pure white color represents those cells where the amount of the first liquid exceeds the amount $x \geq K$ providing the equilibrium between gravity and surface tension.

The postprocessor uses the following library functions:

- CAT_InitPostprocessor

  This function takes as input a binary file containing information about the cellular array.

  The function reads the contents of the cellular array from the file and places the read array into memory.

- CAT_GetCell

  This function takes as input a pointer to a structure and the cell indices in the array.

  It writes the current state of a cell in the array by its indices to the passed structure.

- CAT_FinalizePostprocessor

  This function frees the memory that was used to store the cellular array.

3. Computational experiments

The general parameters of the experiments performed with the synchronous and asynchronous models were as follows:

- The size of the array is 100 by 100 cells.
- The amount of the first liquid in a cell with equilibrium between gravity and surface tension is taken as $K = 100$.
- At the initial time, in each cell a random integer amount of the liquid x ranges from 1 to 350, with a probability of 25 %, otherwise the state value of that cell is $x = 0$.

3.1. Synchronous model results. Figure 1 shows the initial distribution of liquids ($T = 0$) and results of simulation at iterations 1, 10, and 100. The cells shown in white are those with the amount of the first liquid exceeding the value $x \geq K$ needed for equilibrium between gravity and surface tension forces.

At the first iteration, the first liquid actively spreads from the cells with the amount of liquid $x > K$ to the neighboring cells. Then, in the following iterations, the liquid collects into small drops, but after a certain number of iterations ($T = 10$ and $T = 100$) the liquid hardly moves at all.

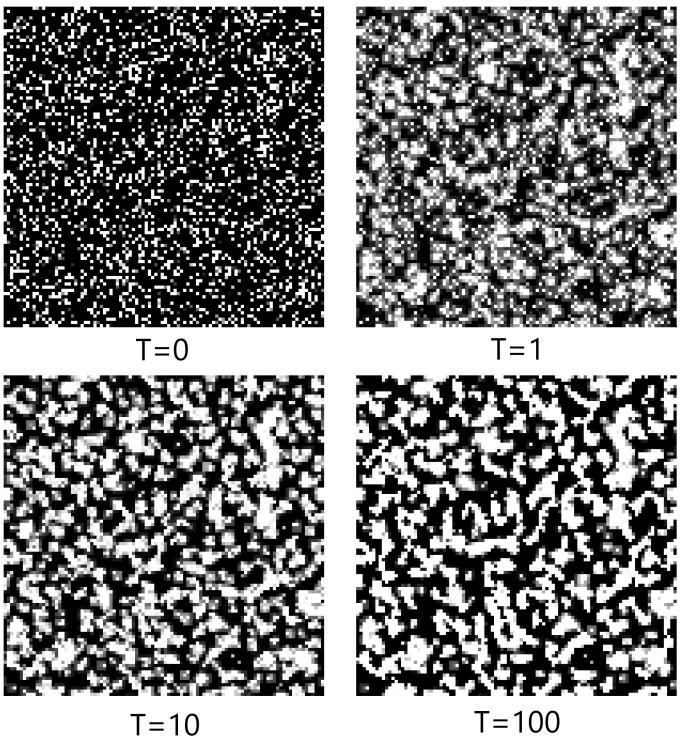

Figure 1. Results of simulation of the phase separation process using the synchronous cellular automaton with integer cell states

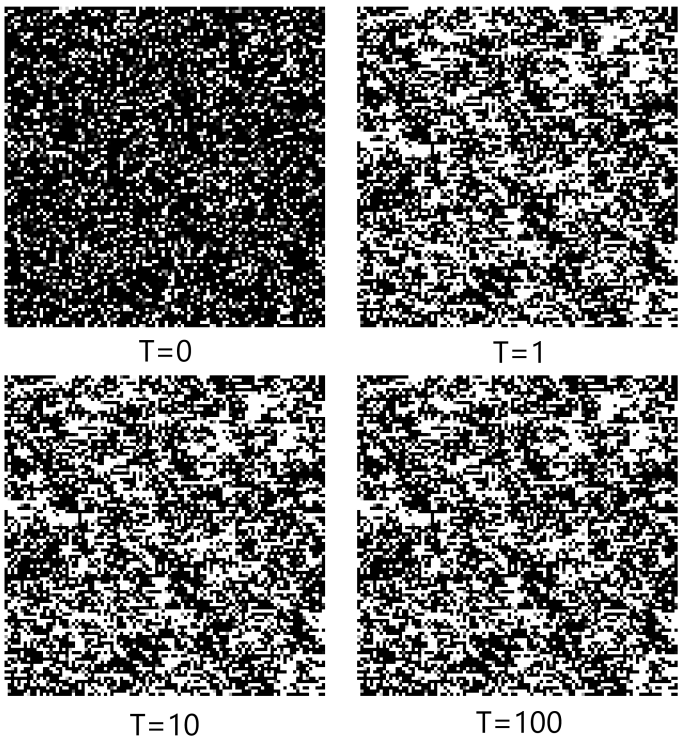

Figure 2. Result of simulation of the phase separation process using the asynchronous cellular automaton with integer cell states

3.2. Asynchronous model results. Figure 2 shows that during the first iteration the first liquid actively spreads from cells with the first liquid amount $x > K$ into the neighboring cells. In the following iterations, the shape and size of the spots remain virtually unchanged. The character of the droplets, as compared to the synchronous model, is more granular and less rounded.

3.3. Comparison with binary model. For comparison, the model of phase separation with a binary alphabet [2] was additionally implemented.

The size of the array was also chosen 100 by 100 cells. Initial states of cells 0 and 1 were distributed equally throughout the array.

In contrast to synchronous and asynchronous models, in the binary model (Figure 3) the droplets are collected in larger spots. Since mass is not conserved in the binary model, depending on the initial conditions, the ratio of liquids can be significantly disturbed over a large number of iterations (up to the complete disappearance during one of them).

Unlike the binary model, the new synchronous and asynchronous models are robust to all initial conditions due to the conservation of liquid mass.

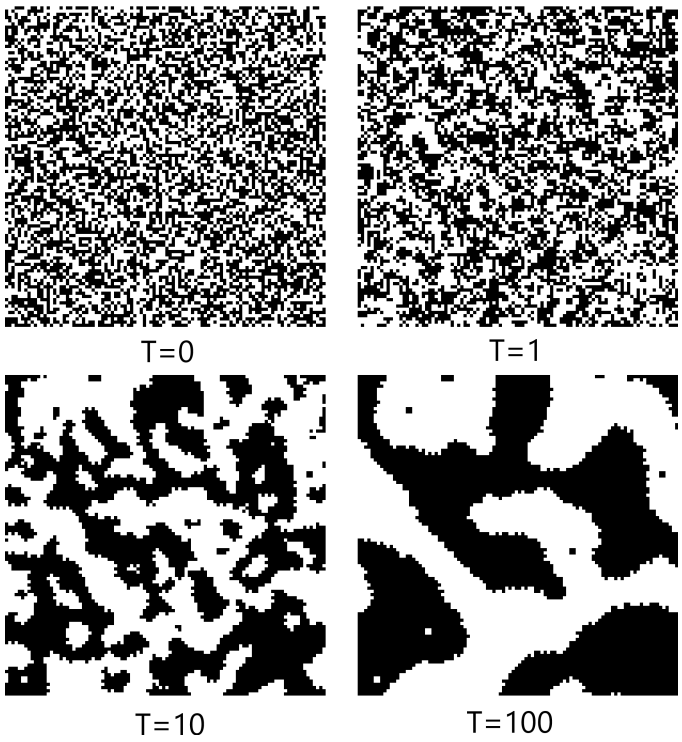

Figure 3. Result of simulation of the phase separation process using the cellular automaton with binary cell states

3.4. Discussion of results. The extent to which the liquid pools into spots depends on physical properties such as viscosity and surface tension force. As a result of the experiments performed for the synchronous and asynchronous models, a qualitative correspondence with the physical view for high viscosity liquids was obtained. Expanding the range towards lower viscosity can be achieved by modifying the transition functions of these models. Thus, larger spots are expected to be obtained.

Based on the above results, it can be concluded that the synchronous model reflects the phase separation process better than the asynchronous model. For the asynchronous model, the high viscosity and the effect of automaton noise make the pattern grainy and bear little resemblance to the real physical view.

Conclusion

During experiments we compared synchronous and asynchronous models and found out that these models match the physical pattern, but they require further improvements.

The advantages of models presented in this paper are the conservation of liquid mass, (due to transition function), and the possibility of controlling the layer thickness, due to the use of an integer alphabet of cell states. The disadvantage of existing state models is that they are suitable only for highly viscous liquids.

In the course of this work, the library of cellular automata typologies was also modified by adding a parallel implementation of the cell array traversal module for shared memory systems for synchronous mode.

Further development of the models presented here is expedient in the direction of describing the phase separation process for liquids with a large viscosity range of the simulated liquids through modifying the cell transition functions.

References

[1] Fridrikhsberg D. Course of Colloid Chemistry. — Leningrad: Chemistry, 1984 (In Russian).

[2] Bandman O. Cellular automata models of spatial dynamics // System Informatics. — Novosibirsk: Publishing House of SB RAS, 2006. — Vol. 10. — P. 59–113 (In Russian).

[3] Medvedev Yu.G. Architecture of the cellular-automata topologies library // Bull. Novosibirsk Comp. Center. Ser. Computer Science. — Novosibirsk: NCC Publisher, 2022. — Iss. 46. — P. 27–41.

[4] Prata S. C Primer Plus: Sixth Edition. — Addison–Wesley, 2013.

[5] PGM Format Specification Available. — https://netpbm.sourceforge.net/doc/pgm.html#index (accessed July 18, 2022).