Complex Systems **10** (1996) 23–41

PECANS: A Parallel Environment for Cellular Automata Modeling

Luigi Carotenuto
Franco Mele
MARS Center,
via Comunale Tavernola,
80144, Capodichino, Napoli, Italy

Mario Mango Furnari
Renata Napolitano
Istituto di Cibernetica, CNR,
via Toiano 6, 80072, Arco Felice (Napoli), Italy

Abstract. This paper discusses a new methodological approach for the modeling of complex physical systems and a parallel programming environment that implements it. The methodological approach is based on a reduction process of a physical phenomenon in components; each component is *represented* in terms of a cellular automaton (CA) and the relations among different components are represented by the CA network abstraction.

To implement this approach, the *Parallel Environment for Cellular Automata Network Simulation* (PECANS) has been developed. The *CANL* language has been defined as a tool to represent models by means of a network of CA. Some examples of modeling and simulations, carried out following the proposed approach, are reported.

1. Introduction

In many cases natural systems interesting from both a fundamental and applicative point of view cannot be easily described in terms of differential equations. In fact, these systems are characterized by a number of different physical parameters which are mutually coupled in a nonlinear way. In any case it is difficult to determine the system evolution and in some cases this representation might not be able to capture the phenomena of interest.

The modeling of these complex systems is difficult due to the necessity of capturing in a single representation the mutual coupling of the different parameters. In many cases this problem can be overcome through a reduction process [1], representing the whole complex phenomenon as composed of a number of simple models, called *model components*. Each model component represents a physical aspect of the phenomenon (e.g., the motion, the chemical reactions, the heat flow, etc.); the different model components are connected in such a way to represent the whole phenomenon. This methodology allows one to overcome the difficulties encountered in the representation of a complex system, as shown by many modeling examples which are based on a sort of abstraction by means of reduction. However, up to now a systematic approach has not yet been proposed for this methodology.

To allow the reduction of the phenomenon and the composition of its components, we have chosen the cellular automata (CA) as a representation for each model component. A CA is a bidimensional grid of identical cells; the changes of cell states are represented, at each time step, by the data variation on the grid according to a *transition rule*; this rule specifies the new state for each cell according to its state and the state of neighbor cells at the previous time step.

In addition, the CA approach offers the following main advantages.

- It is strongly expressive since the model description is attained by means of local rules, which also allows the implementation of qualitative aspects of the model.
- The computational model is intrinsically parallel.

According to these methodological choices, we propose to represent a phenomenon by a set of model components, each of them being a CA, connected in a network: we have named this representation *cellular automata network*.

In order to apply this methodology we have developed a *Parallel Environment for Cellular Automata Network Simulation* (PECANS); this environment allows the parallel execution of physical phenomena simulations by means of a CA network. One of the main objectives was the definition of a new programming language, specifically dedicated to implement the proposed approach.

In fact, the languages of the qualitative physics as in [2, 3, 4] provide an interesting methodological approach to carry out modeling through model compositions. However, these languages do not result in an adequate representation for complex physical systems [3, 4]. So we introduced the model compositions in the context of CA modeling.

The developed *Cellular Automata Network Language*, (CANL), consists mainly of the following.

- A set of constructs to describe the components of the whole model.
- A composition mechanism for such components.

An important feature of CANL is the fact that it is functional oriented, although the possibility to be easily cross-compiled into a standard programming language, the C language, is retained. Furthermore, with respect to the standard CA model, we also added the possibility of defining global variables as well as "real" variables relevant to the modeled physical phenomenon. However, these extensions did not stimulate us to define a new name for the *structure* (form or formalism, etc.) which represents a model component: we continued to call CA this sort of computational metaphor.

In PECANS two main kinds of parallelism are possible. The first one is the *data* parallelism intrinsic to the CA computational model, because the CA computing rules are essentially local. The second one is the *control* parallelism form which concerns the network of CAs, while the data parallelism concerns each model component.

In dealing with parallelism we tried to move the complexity of managing it from the user to the system developer, designing the CANL language without annotation for parallelism; in this way the parallel programming details are masked to the user.

In the following we examine the modeling of aggregation, to give an example of a reduction process in model components and their interconnections. The rest of the paper is organized as follows. In section 2 the whole structure of the PECANS environment is described. In section 3, the CANL language and its features are described. In section 4, two examples of using PECANS to simulate some physical phenomena are given. Finally, in section 5 a comparison with other systems and the current implementation status are reported.

1.1 A cellular automata network example for aggregation phenomenon

In Figure 1 we report an example of net abstraction that models a simulation of the colloidal aggregation phenomenon [5]. The sensory analysis [1] of the physical system revealed an ontological-molecular behavior, that is, the physical phenomena involved can be locally described by means of a set of interaction rules among primitive components—cells or molecules. The individual state of each cell is defined by a set of parameters, and its updating is given by a local rule that takes into account the state of the cell together with the states of the cells belonging to its neighborhood, providing in such a way a microscopic description. The time evolution of the states of the cells determines the global state of the system approaching its macroscopic description. For this reason the chosen representation is based on a cellular automata network.

The principal processes we consider are: the motion of single particles (monomers) and clusters; the mutual aggregations of monomers or the aggregation of a monomer to a previously formed cluster, and the interactions among clusters. The modeling of these processes involves the following eight automata.

C1 Determination of interacting particles or clusters and cluster formation.

C2, C3 Determination of particle and cluster masses.

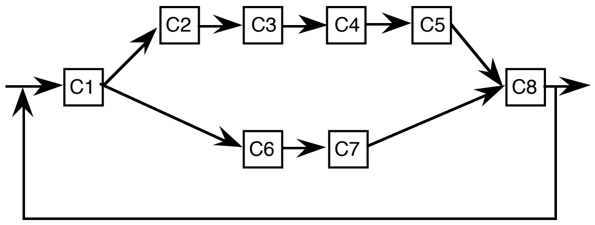


Figure 1: Cellular automata network for aggregation example.

C4, C5 Determination of particle and cluster mobility.

C6, C7 Determination of particle and cluster motion direction.

C8 Application of displacement rules, connected as shown in Figure 1.

In our approach, each different aspect of a physical system, for example, particle aggregation, motion, and collisions, is described by a specific model component. The whole physical process is simulated by means of a network of CA.

In our environment powerful linguistic primitives are available; they describe the dynamic properties of individual particles or clusters of physical systems. Particular attention is given to the coupling of single automata (components) that are connected by means of a network.

A CA simulator must allow us to carry out simulation studies within a reasonable amount of computation time. In other words, the main interest for developing a simulation environment consists in the possibility of simulating CA networks so complex to allow capturing the most interesting aspects of physical phenomena.

2. The parallel environment for cellular automata network simulation

When developing PECANS the first goal was to maintain the user-experienced complexity as low as possible, in particular with respect to the parallel programming idiosyncrasy. To attain this goal we adopted a modular programming methodology; in this way, the user can concentrate efforts on describing physical problems using the CANL language. In other words, the user will be involved only with the complexity of simulating the physical problem and will not be overloaded with the intricacy of the computing system.

User requirements we have tried to meet are user friendliness, and the availability of an efficient and expressive programming language to represent in a clear and simple way the model component computation.

To simplify the user interactions with PECANS we developed a set of graphic oriented tools. To reduce the programming complexity we defined a functional-like programming language, where the functional composition is the main control structure to compose the behavior of the CA components into the global behavior of the CA network.

To attain the goal of efficiency the language is cross-compiled in **C**, and to deal with the CA modeling intrinsic parallelism we placed its management partially in the cross-compiler and partially into the run-time system. The parallelism is managed by the cross-compiler in the sense that it will produce an annotated code that will be compiled and linked with parallel libraries. Then, the resulting program will run on different parallel architectures, ranging from vectorial machines, to parallel machines with shared or distributed memory. The idiosyncracies of these different kinds of architecture are managed by the cross-compiler and the run-time system.

The global structure of PECANS is built around the following four subsystems.

- *User interface.* It is graphic based and it is implemented on top of the X-window system. There is a text editor to write the simulation programs and a graphic interface to look at CA network structures and simulation results.
- *CANL.* It is used to program CA networks. It is a functional-like oriented language, allowing a hierarchical description of problem structures.
- *Cross compiler.* It accepts as input a CANL program and produces as output a C program with the parallel opportunities annotated for different parallel machine architectures. This C program is then compiled and linked to the simulator with the corresponding parallel run-time systems.
- *Kernel of the parallel run-time.* It is implemented on a parallel machine. It accepts the CANL cross-compiled programs and executes them on a parallel machine, returning the results to the graphic user interface.

The whole structure of the parallel programming environment for the CA network is sketched in Figure 2.

3. The CANL language

CANL furnishes the possibility to describe model components and connections among them. In CANL each component is mainly represented by an automaton grid and by a transition function that describes the evolution of the physical system from the state at the step n to the state at the step $n+1$. Furthermore, CANL allows the definition of a net for the connections of the components and expressing in a clear and simple way the model component computation. To allow users to attain some kind of inter-CA communications, the use of global variables and the **etfai** operator were introduced into CANL.

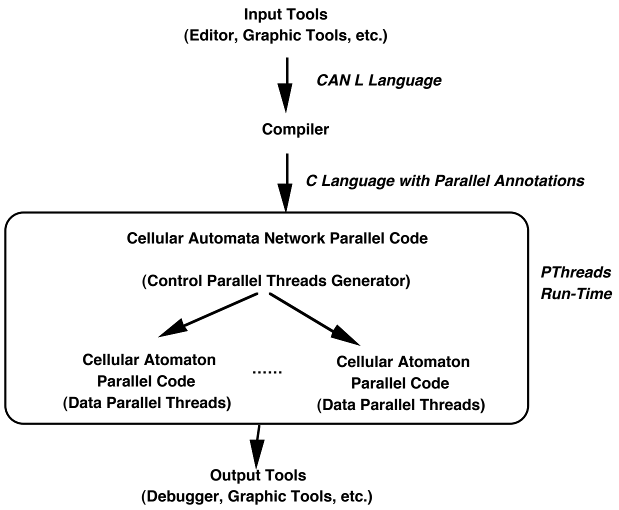


Figure 2: Parallel simulation environment for CA network structure.

3.1 Cellular automata network definition

A network of CA is represented in CANL by a graph, where each node represents an automaton, and an edge a precedence relation. Each automaton is denoted by a *name*, and its behavior is described by a set of *properties*, a *transition function*, and a *neighborhood type*. The property of an automaton represents a particular physical quantity to be simulated.

Let A and B be two automata, if one or more properties of the automaton A are used inside the transition function of the automaton B, then we say that B *depends on* A. Each automaton is the *owner* of the properties defined along with it; the writing right on a property is guaranteed only to the owner of that property, while the reading right is granted to all automata.

At each computational step, the transition functions of all the automata compounding the network will be executed. According to the precedence relations, a network automaton can be executed only if the executions of the antecedent automata in the network have terminated its computational step.

In Figure 3 the primitive function `def-net` used to define the network of CA is shown.

```
(def-net (:no-dip <automata-list>)
         (:dip <automaton> <automata-list>)
                ...
         (:dip <automaton> <automata-list>)
)
```

Figure 3: The primitive function `def-net`.

The expression (`:no-dip` <automata-list>) takes into account all the automata that do not have any dependence with other automata for each step. Nevertheless these automata can refer to other automaton property values computed at the previous time step.

The values of a property for each single component correspond to a physical state of the modeled system only at the end of a computational step for the whole network.

In the following an example of the primitive function `def-net`, regarding the CA network simulating the colloidal aggregation phenomenon shown in Figure 1, is reported.

```
(def-net    (:no-dip (C1))
            (:dip C2 (C1)) (:dip C3 (C2))
            (:dip C4 (C3)) (:dip C5 (C4))
            (:dip C6 (C5)) (:dip C7 (C6))
            (:dip C8 (C5 C7)))
```

3.2 Global variables definition

In making a model of a physical system it is often necessary (and useful) to describe some global features of the system (for example the total mass of a cluster of particles). We decided to introduce the possibility of defining global features in CANL.

A global feature is represented in CANL by a *global variable*; it is defined along with a transition function, whose execution will modify its value. Its value can be modified during the execution of the transition function of each cell of the automaton. This value is then available (only in reading mode) to the successive automata in the network.

It is possible to define two types of global variables: scalar variables and vector variables. Their definitions are contained in the slot `:glob-variables` of the primitive operator `def-transition`.

As an example we report the transition functions relative to the dendritic crystal example described in section 4.2. The global vectorial variable `temp-mass` is defined and modified in the transition function `count`:

```
(def-transition
   :function-name      count
   :glob-variables     ((vector-def temp-mass int 1000 0))
   :body               (if    (neq(center label_n) 0)
                       (then
                            (set (vector temp-mass (center label_n))
                                 (+  (vector temp_mass (center label_n))
                                     (centerparticles_mass)))))
                            )
```

`count` is used, in reading mode, in the transition function `mass_sum` of a successive automaton of the network:

```
(def-transition
  :function-name     mass_sum
  :glob-variables    ()
  :body              (if   (not(eq (center label_n) 0))
                     (then
                                (set (center_w cluster_m)
                                     (vector temp-mass(center label))))
                     (else     (set (center_w cluster_m) 0))))
```

3.3 The control construct `etfai`

In a network it may happen that the transition function has to be applied more than one time on the same automaton of the network, before the computation of the successor automata (according to the precedence relations) starts. The number of times the transition function is executed on the same automaton, depends on a given condition.

To meet this requirement, we introduced the construct `execute-transition-function-again-if` (`etfai`), that can be called inside the transition function. This construct takes one or more arguments: the first one is an exit condition for the iterations; the other ones are optional and may contain, for each property of the automaton, an expression to initialize the values of a property to the property's values of an automaton which the first one depends on. The exit condition is computed locally for each cell. To have the transition function applied more than one time on all the cells of the automaton, it is sufficient for the condition to be verified at least on one cell, during the application of the transition function. In this respect the `etfai` construct differs from all the other CANL constructs.

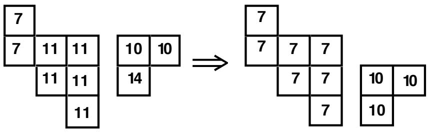
To show the use of the `etfai` construct we report in Figure 4 the code that implements particle labeling in a dendritic crystal example, described in section 4.2.

We have used an algorithm where each cell is compared with the neighborhood cells, and its value is updated to the smallest value among the neighborhood cells. This procedure is repeatedly executed until no more updating occurs.

In this example, since the automaton that carries out the labeling process belongs to a CA network, the labeling must be totally executed before executing the other automata in the network; this is the reason why we introduced the `etfai` construct as a control mechanism.

3.4 Model component definition

In CANL each model component is described by an automaton. The automaton is denoted by a *name* and its behavior is described by a set (possibly empty) of *properties*, by a *transition function*, and by a *neighborhood type*. The properties can correspond to physical quantities (e.g., temperature or mass) or to some features (e.g., the probability of a particle to move itself).



```
(def-transition
  :function-name make_label
  :global-variables ()
  :body
      ((if (not (eq(center label) 0))
       (then (set  (center_w label)
                   (min_neighbor label)))
       (else (set  (center_w label)
                   (center label))))
   (etfai
      (not (eq     (center label)
                   (center_wlabel)))
      (init labelold-label)))
```

Figure 4: An example of `etfai` construct for the labeling algorithm.

The values of a property can also be real values. Anyway, according to the CA model, a property corresponds to a computational grid as in [6].

To take account of the CA structure, we defined a new data structure, the *automaton grid*, and a set of operators to access its elements. These operators have been defined as generic operators, so the user can refer to the cell neighbors by simply invoking them on the property's name without having to specify a particular neighbor cell position. For example, the user can refer to the neighborhood elements of each cell of the property named `temperature` by invoking `(centertemperature)`, `(northtemperature)`, or `(center_wtemperature)`. The operator in the last expression is used for writing operations, whereas the previous ones are used only for reading operations.

The transition function is built around a set of primitives and user defined functions, called *auxiliary functions*. The current neighborhood types implemented are *North East West South* (NEWS), the *eight-neighborhood*, and the *hexagonal neighborhood*.

For each transition function, the *border conditions* to be used when the transition function is applied to the grid border cells, has to be specified. Currently two types of boundary conditions are implemented; the *toroidal* one, where the grid is closed on itself, and the *adiabatic* one, where the missing neighbor cell is assumed to be the same as neighborhood center. For each transition function *constants* can be used in the transition function body.

The CANL set of primitive functions for the model component are as follows.

`def-automaton`, used to define an automaton of the network;

`def-transition`, used to define the automata transition function;

```
               (def-automaton
                  :name                         <name>
                  :neighbourhood-type           <neighbourhood-type>
                  :dimension                    <dimension>
                  :border-conditions            <border-conditions>
                  :constants                    <constants>
                  :transition-function-name     <transition-function-name>
                  :list-of-properties           <list-of-properties>
)


(def-transition
  :function-name    <function-name>
  :glob-variables   <glob-variables>
  :body             <body>
)


(initialize
  :automaton-name   <automaton-name>
  :property-name    <property-name>
  :body             <body>
)

(def-aux-fun
  :function-name       <function-name>
  :type                <type>
  :neighbourhood-type  <neighbourhood-type>
  :parameters-list     <parameters-list>
  :body                <body>
)
```

Figure 5: The automaton data structure and functions syntax.

`def-aux-fun`, used to define the auxiliary functions that may be called inside the transition functions of all the automata in the network;

`initialize`, used to initialize the values of a property at the beginning of a simulation.

See Figure 5 for their syntax.

The <body> of `def-function` and `def-aux-fun` is constructed by using the functional composition mechanism and some arithmetic and logical primitive operators, together with the assignment and the conditional operators.

To demonstrate the expressiveness and the simplicity of the CANL language, we show the main functions of the CANL program for the well-known game LIFE [6]. LIFE may be thought of as describing a population of stylized organisms, developing in time under the effect of counteracting propagation and extinction tendencies.

The transition function is built in the following way.

- A live cell, represented by the grid cell value **1**, will remain alive only when surrounded by either 2 or 3 live neighbors; otherwise, it will feel either "overcrowded" or "too lonely" and it will die.

- A dead cell, represented by the grid cell value **0**, will come to life when surrounded by exactly 3 live neighbors.

We first define the automaton, named `life`, using the following function.

```
(def-automaton
   :name                        life
   :neighborhood-type           eight
   :dimension                   256
   :border-conditions           toroidal
   :constants                   ()
   :transition-function-name    life-fun
   :list-of-properties          ((organisms int))
)
```

Next, we define the auxiliary function `neighbors_sum` to calculate the sum of all neighbors as follows:

```
(def-aux-fun
   :function-name          neighbors_sum
   :type                   int
   :neighborhood-type      eight
   :parameters-list        (((property prop) int))
   :body                   (+(north prop) (east prop)
                              (south prop) (west prop)
                              (north-east prop) (north-west prop)
                              (south-east prop) (south-west prop)
                           )
)
```

where `property` is a keyword, used for identifying properties' names. Then, we define the transition function `life-fun` for the automaton as follows:

```
(def-transition
   :function-name      life-fun
   :glob-variables     ()
   :body               (case (center organisms)
                          (1   (case (neighbors_sum organisms)
                               ((23) (set (center_w organisms)
                                          (center organisms)))
                                  (otherwise (set (center_w organisms)
                                          0))))
                          (0   (case (neighbors_sumorganisms)
                               (3      (set (center_w organisms)
                                            1))
                               (otherwise   (set (center_w organisms)
                                            (centerorganisms)))))
                       )
)
```

4. Simulation examples in CANL

In this section we describe two experiments carried out to validate the approach adopted for modeling physical systems. The first example is about the simulation of the interaction of bubbles and drops with a solidification front in a microgravity setting (section 4.1). The second example is about the aggregation of small particles to form a large cluster, and how to use a network of CA to simulate such a complex system (section 4.2).

4.1 Melting and solidification

This simulation was carried out in preparation for an experiment executed in microgravity during the Spacelab mission IML-2 (July 1994). The experiment intended to study the interaction of bubbles and drops with a solidification front. The experiment set-up consisted of a solid sample contained in a closed cell, to be made molten and then solidified in a temperature gradient. The performed simulation was aimed at determining the behavior of the sample under different thermal conditions (temperature gradients, heat exchange through the boundary walls); the results provided information needed to design the experimental cell flown on board the Spacelab.

The two-dimensional model system considered is described as follows. The sample is represented by a matrix of elements (cells of a CA) bounded by two lateral walls and two plates. The cell status is represented by two (real values) properties: the temperature and the phase, defined as the percentage of solid phase content. Each cell exchanges heat by diffusion with its neighbors; at each step the heat exchanged is computed to determine the new temperature. If the temperature evolution crosses the melting point, the phase transition is modeled as follows: part of the computed heat exchanged is needed to reach the melting point; the remaining part of the heat exchanged is available for phase transition; then the corresponding phase variation can be evaluated. Once the transition is completed, the cell temperature can change again according to the Fourier law. In the following we report the definition of the automaton.

```
(def-automaton
   :name                     melting_sol
   :neighborhood-type        4
   :dimension                100
   :border-conditions        adiabatic
   :constants                ((solid 1)     (liquid 0)
                             (Tm0 51)      (densi_l 0.7682)
                             (densi_s 0.796)      (Kl 0.001525)
                             (Ks 0.00158)      (Cl2.512)
                             (Cs 2.303)      (Q 251.43)
                             (alpha_l(/ Kl (* Cl densi_l)))
                             (alpha_s (/ Ks (* Csdensi_s)))
                             )
```

```
    :transition-function-name    transition
    :list-of-properties          ((temperature double)
                                 (phase double)
                                 (containerint)
                                 )
   )
```

The transition function action can be summarized by these lines:

```
if the cell grid corresponds to the container
then do nothing
else  if the cell is solid or liquid
      then    compute the new temperature (according to the
                  Fourier law)
              if the transition is started
              then compute the new phase
              else the phase is the same (solid or liquid)
      else    (it is in the transition phase)
              compute the new phase,
              if the transition is terminated,
              then compute the new temperature
              else the temperature is the melting point.
```

The simulation has been validated by comparing the results for a one-dimensional case with the corresponding analytic solution. A temperature difference was imposed between the plates, at constant and uniform temperatures of, respectively, T_0 and $T_0+\Delta T$; adiabatic conditions were imposed on the lateral walls; the sample was assumed to be pure liquid tetracosane (the model substance used in the Spacelab experiment); the tetracosane melting point is 51°C; its initial temperature was uniform and equal to $T_0 + \Delta T$.

Figure 6 shows the temperature profile in the sample after 75 simulated minutes; the position of the solidification front is indicated by the edge. The agreement with the analytic solution is completely satisfactory.

4.2 Dendritic crystal started from multiple cell-seeds (each cell is a potential seed)

The understanding of the aggregation of small particles to form large clusters is fundamental for a number of scientific and technological fields. The main mechanisms of aggregation are based on particle-cluster and cluster-cluster interaction. The relative importance of these mechanisms is influenced by the dynamics of the suspension. Simulations performed up to now generally do not consider all the mechanisms involved in the aggregation. Many studies have been conducted on the Witten–Sander model [7], based on the particle-cluster interaction: single particles diffuse towards a fixed seed, where the cluster grows. This process has also been simulated with CA [6].

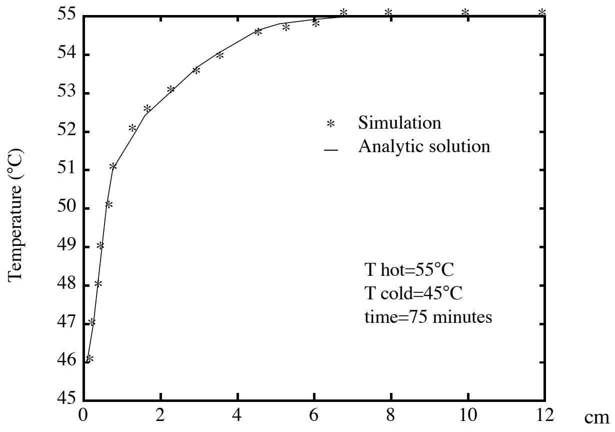


Figure 6: Comparison between CANL simulation of solidification and analytical solution.

Other simulations have been conducted considering cluster-cluster interaction using a Monte Carlo approach [8]: larger clusters were formed by sticking together two smaller clusters with a random relative orientation.

The goal of our simulation is to model all the different interaction mechanisms together and to determine their dependence on the dynamic conditions. Our methodological approach allows the representation of each physical aspect as a model component. Both particles and clusters move according to the considered model (e.g., free diffusion, diffusion and sedimentation, etc.) and interact when they are in contact. A sticking probability can be introduced to take into account the case of a potential barrier slowing down the aggregation rate.

In order to execute this simulation, we defined an eight cellular automata network, whose structure is sketched in Figure 7.

The network structure definition by the `def-net` operator is as follows.

```
(def-net
  (:no-dip (LABEL))
  (:dip DIRECTION (LABEL))
  (:dip CLUSTER_DIRECTION (DIRECTION))
  (:dip MASS_COUNT (LABEL))
  (:dip MASS_CLUSTER (MASS_COUNT))
  (:dip MAYBE_MOVE (MASS_CLUSTER))
  (:dip MAYBE_MOVE_CL (MAYBE_MOVE))
  (:dip MOVE (CLUSTER_DIRECTION MAYBE_MOVE_CL))
)
```

In the following we report the task of each automaton.

LABEL This automaton determines the interacting particles and/or clusters, making the labeling for the new clusters.

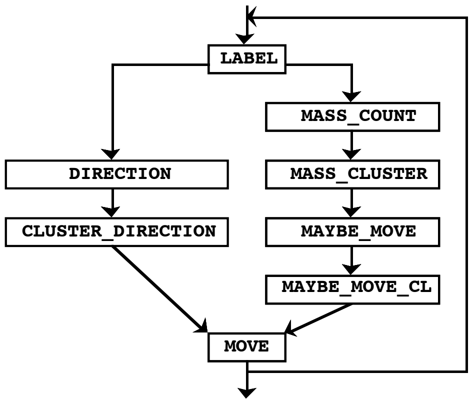


Figure 7: Automata network structure for dendritic crystal simulation.

MASS_COUNT and **MASS_CLUSTER** These two automata determine the cluster total mass, where the cluster mass is given by summing the mass of each particle belonging to the cluster.

DIRECTION and **CLUSTER_DIRECTION** These two automata assign to each particle or cluster the motion direction. The first automaton finds a direction for each particle according to the considered model (for instance, using a random number generator, built in the system, in case of Brownian motion), while the second automaton ensures that the direction is the same for all the particles belonging to the same cluster.

MAYBE_MOVE and **MAYBE_MOVE_CL** These two automata determine at each step if a particle or a cluster will move. In general, the relative mobility of clusters depends on cluster mass, according to the conditions being modeled. For instance, under microgravity conditions the probability to move diminishes when the cluster mass increases, while under normal gravity this probability increases with the cluster mass in the gravity direction.

MOVE This automaton moves the particles and the clusters, taking into account their directions and probabilities of movement. If, according to these directives, two or more particles want to occupy the same place, then in this place the new particle will have a mass given by summing the mass of all these particles.

As an example, we report in Figure 8(a) and 8(b) two aggregations simulated with a 400×400 grid; in both cases the initial condition is given by a uniform distribution of particles, scattered randomly in the plane. In Figure 8(a) the cluster mobility increases with the mass; this case is similar to the Witten–Sander model, but the clusters are not symmetric because no

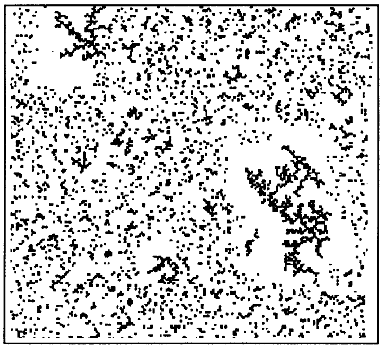


Figure 8(a): Aggregation with cluster mobility increasing with cluster mass.

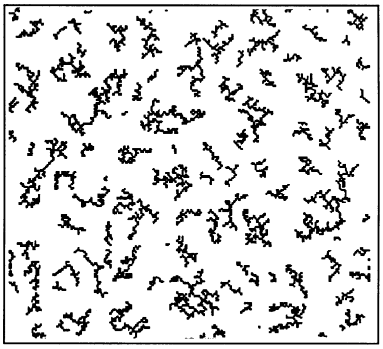


Figure 8(b): Aggregation with cluster mobility decreasing with cluster mass.

seed is present. In Figure 8(b) free diffusion has been modeled: the cluster mobility decreases with the mass. This condition is expected for microgravity experiments.

5. Some comparisons and conclusions

Our aim was to propose a new methodological approach for the modeling of systems characterized by a molecular ontology. This approach is essentially based on creating physical models by means of a reduction process in model components. To this end we have extended the CA model to a network of CA, where each automaton can have an unlimited number of properties. Each property of an automaton can have real values, and it is possible to use global variables and the control construct `etfai`. We have defined the PECANS environment and the CANL language as tools to apply this approach.

Our prerogative has been to develop an user-friendly programming environment and a language with good expressiveness and simplicity in its use.

In this respect, it may also be possible to capture some qualitative physical aspects.

The environment is implemented to run on a parallel machine. Nevertheless, no overhead for parallelism is burdened on the programmer; everything is automatically done by the environment. In fact, PECANS can also run on a sequential architecture, with no perceivable modification for the CANL programmer, except for time performance.

Other CA environments have been proposed in the last few years. The CAM-6 machine [6] is perhaps the only way to deal with a language specifically dedicated to CA that allows the programmer to be unaware of the implementation details. But the high efficiency is due to the hardware implementation, and this causes constraints in its use. In fact, it is suitable only for representing CA with a number of states not higher than sixteen, being the number of possible values of a single cell and the number of computational planes limited by the hardware constraints.

The CAMEL [9] environment is implemented on a transputer network, and there is no practical limit to the number of states for the elementary automaton; this fact is one of the motivations that make it suitable for realizing interesting simulations. It also has the efficiency features, since it is implemented on a transputer network, with a load balancing strategy to introduce time optimization, without charging the user for that. However, there is no real programming language to make the environment implementation transparent to the user.

The CAPE [10] parallel environment is also implemented on a transputer network, in which no overhead for parallelism is charged to the user. The same user program can be executed either sequentially or concurrently. But the user program has to be written in FORTRAN, and the number of states that an automaton cell can have is limited.

In [11] a shared MIMD machine is used, as in our implementation, and data parallelism is achieved by dividing the cell grid updating among the available processors. But there is no language for CA programming.

We conclude this paper by reporting the current implementation status of PECANS, and how we intend to proceed in our work.

At present, the kernel of our environment implements essentially the data parallelism intrinsic to each automaton. The system runs on parallel machines; the automaton grid is split according to its topology and each partition is associated to a thread. We observed that the speed-up factor attains the limit values on increasing the transition function complexity, and it is essentially independent of the number of times the transition functions are iterated, but it is strongly dependent on the automaton grid size [12].

At present, we are working on tuning the implementation of our run-time system for different parallel architectures; this involves only a few modifications to our program, because it is written in a modular way. Obviously the CANL programs will be the same for all the architectures; the only thing to do is to specify the architecture type during the environment installation.

This feature is very important, because it allows the system to be portable to different architectures.

We are also working on control parallelism, that is, to produce the code to run more than one automaton concurrently according to the precedence relations described by the network of CA.

Acknowledgements

The authors are grateful to Professor S. Di Gregorio not only for the useful discussions and suggestions, but mainly for his encouragements. However, he is not responsible for the contents of this paper.

This study has been conducted under the sponsorship of the Italian Space Agency.

References

[1] P. A. Fishwick, "Process Abstraction in Simulation Modeling," in *Artificial Intelligence, Simulation, and Modeling*, edited by L. E. Widman *et al.* (Wiley Interscience, 1989).

[2] B. Kuipers, "Qualitative Simulation," *Artificial Intelligence*, **29** (1986) 289–338.

[3] S. Bandini, G. Cattaneo, S. Cordioli, and G. Vian, *A Qualitative Description of Field Theory in Terms of Molecular Ontology*, Proceedings 1°National Conference of Italian Association of Artificial Intelligent, Trento, (1989).

[4] P. Hayes "Naive Physics I: Ontology for Liquids," in *Formal Theory of the Commonsense World* by J. R. Hobbs and R. C. Moore (Ablex Pub. Co., NJ, 1985).

[5] L. Carotenuto and F. Mele, *Simulation of fractal aggregation in microgravity using cellular automata approach*, Forty-second Congress of the International Astronautical Federation, Montreal (1991).

[6] T. Toffoli and N. Margolus, *Cellular Automata Machines: A New Environment for Modeling*, (MIT Press, 1987).

[7] T. A. Witten, L. M. Sander, *Physical Review Letters* **47** (1981) 1400.

[8] P. Meakin, *Journal of Physics A*, **20** (1987) L1113.

[9] D. Barca, G.M. Crisci, S. Di Gregorio, and F. Nicoletta, "Cellular Automata for Simulating Lava Flows: A Method and Examples of the Etnean Eruptions," *Transport Theory and Statistical Physics*, **23** (1994) 195–232.

[10] M. White and M. G. Norman, *CAPE Cellular Automata and Beyond ...*, Technical Report ECSP-TN-39, Edinburgh Parallel Computing Center (1989).

[11] M. Tomassini, "Cellular Automata Calculations on a Shared Memory Mimd Machine," *Supercomputing Review*, November, 1990.

[12] M. Mango Furnari, F. Mele, and R. Napolitano, “A Parallel Environment for Cellular Automata Network Simulation,” in *Second Workshop on Massive Parallelism*, edited by M. Mango Furnari (Capri, Italy October 3–7, 1994, World Scientific Press).