# SIMULATION OF NATURAL PHENOMENA BY CELLULAR AUTOMATA WITH THE LIBAUTOTI LIBRARY: AN APPLICATION TO GEOLOGICAL MODELLING

**William Spataro[(a)], Giuseppe Spingola[(b)], Giuseppe Zito[(c)], Donato D'Ambrosio[(d)], Rocco Rongo[(e)], Maria Vittoria Avolio[(f)] , Salvatore Di Gregorio[(g)]**

[(a) (b) (c) (d) (f) (g)]Department of Mathematics and HPCC, University of Calabria, Italy
[(e)] Department of Earth Sciences and HPCC, University of Calabria, Italy

[(a)]spataro@unical.it, [(b)]g.spingola@gmail.com, [(c)]giuseppe.zito@gmail.com, [(d)]d.dambrosio@unical.it, [(e)]rongo@unical.it, [(f)]avoliomv@unical.it, [(g)]dig@unical.it,

**ABSTRACT**
This work presents the application of libAuToti, an open-source parallel library for implementing models based on the Cellular Automata approach, for the simulation of geological processes. In particular, we describe the implementation of two models for lava flows and debris flows, as defined by the SCIARA and SCIDDICA models, respectively. Cellular Automata are parallel computing models which are continuously gaining attention from the Scientific Community for their potentiality and efficiency. libAuToti permits a straightforward and simple implementation of Macroscopic Cellular Automata models, which are appropriate for the simulation of spatial extended dynamical systems. Experiments have demonstrated the elevated computational efficiency of the library, executed both on an HPC machine and a standard multi-core PC, confirming the reliability of the library and goodness of simulation results.

Keywords: natural phenomena simulation, cellular automata, lava flows, debris flows, parallel computing

## 1. INTRODUCTION

Cellular Automata (CA) are discrete dynamical systems, widely utilized for modelling and simulating complex systems, whose evolution can be described in terms of local interactions. Well known examples are Lattice Gas Automata and Lattice Boltzmann models (Succi 2004), which are particularly suitable for modelling fluid dynamics at a microscopic scale. However, many natural phenomena are difficult to be modelled at such scale, as they generally evolve on very large areas, thus needing a macroscopic level of description. Moreover, they may be also difficult to be modelled through standard approaches, such as differential equations (McBirney and Murase 1984), and Macroscopic Cellular Automata (MCA) (Di Gregorio and Serra 1999) can represent a valid alternative.

Among the above mentioned phenomena, lava flows and debris flows may involve serious dangers for people security and property, and their forecasting could significantly decrease these hazards, for instance by simulating paths and evaluating the effects of control works (e.g. embankments or channels).

As regards MCA, *libAuToti* (Spataro et al. 2008) has proven to be an efficient and flexible simulation parallel library. Written in C++, it allows for a simple and concise definition of both the transition function and the other characteristics of the MCA model definition. Moreover it was projected for both sequential and parallel execution, both on shared and distributed memory machines (thanks to the adoption of the Message Passing paradigm for the inter-processes communications), by completely hiding parallel implementation issues to the user.

In the following, MCA are briefly presented and the main characteristics of the *libAuToti* library illustrated. Two examples of application are also shown, which regard the implementation of the MCA basic lava flow model SCIARA and the MCA landslide model SCIDDICA version "T". A general discussion on the results and on the future perspective of the library concludes the paper.

## 2. MACROSCOPIC CELLULAR AUTOMATA: THE SCIARA AND SCIDDICA MODELS

As previously stated, CA are dynamical systems, discrete in space and time. They can be thought as a regular $n$-dimensional lattice of sites or, equivalently, as an $n$-dimensional space (called cellular space) partitioned in cells of uniform size (e.g. square or hexagonal for $n$=2), each one embedding an identical finite automaton. The cell state changes by means of the finite automaton transition function, which defines local rules of evolution for the system, and is applied to *each* cell of the CA space at discrete time steps. The states of neighbouring cells (which usually includes the central cell) constitute the cell input. The CA initial configuration is defined by the finite automata states at time $t$=0. The global behaviour of the system emerges, step by step, as a consequence of the simultaneous application of the transition function to each cell of the cellular space.

When dealing with the modelling of spatial extended dynamical systems, MCA can represent a valid choice especially if their dynamics can be

described in terms of local interaction at macroscopic level. Well known examples of successful applications of MCA include the simulation of lava (Crisci et al. 2004) and debris flows (Di Gregorio et al. 1999) forest fires (Trunfio 2004), agent based social processes (Di Gregorio et al. 2001) and highway traffic (Di Gregorio et al. 1996), besides many others.

By extending the classic definition of Homogeneous CA, MCA facilitate the definition of several aspects considered relevant for the correct simulation of the complex systems to be modelled. In particular, MCA provide the possibility to "decompose" the CA cell state in "substates" and to allow the definition of "global parameters". Moreover, the dynamics of MCA models (especially those developed for the simulation of complex macroscopic physical systems such as debris or lava flows) is often "guided" by the "Minimisation Algorithm of the Differences" (cf. Di Gregorio and Serra 1999), which translates in algorithmic terms the general principle for which natural systems leads towards a situation of equilibrium. Refer to Di Gregorio and Serra (1999), for a complete description of the algorithm, besides theorems and applications.

### 2.1. The MCA lava flow model SCIARA

SCIARA is a family of bi-dimensional MCA lava flow models, successfully applied to the simulation of many real cases, such as the 2001 Mt. Etna (Italy) Nicolosi lava flow (Crisci et al. 2004), the 1991 Valle del Bove (Italy) lava event (Barca et al. 1993) which occurred on the same volcano and employed for risk mitigation (D'Ambrosio et al. 2006). In this work, the basic version of SCIARA (Barca et al. 1993) was considered and its application to the 2001 Nicolosi (Sicily) event shown.

SCIARA considers the surface over which the phenomenon evolves as subdivided in square cells of uniform size. Each cell changes its state by means of the transition function, which takes as input the state of the cells belonging to the von Neumann neighbourhood. It is formally defined as

$$SCIARA = < R, X, Q , P, \sigma >$$

where:

- $R$ is the set of points, with integer coordinates, which defines the 2-dimensional cellular space over which the phenomenon evolves. The generic cell in $R$ is individuated by means of a couple of integer coordinates $(i, j)$, where $0 \leq i < i_{max}$ and $0 \leq j < j_{max}$.
- $X = \{(0,0), (0, -1), (1, 0), (-1, 0), (0, 1)\}$ is the von Neumann neighbourhood relation, a geometrical pattern which identifies the cells influencing the state transition of the central cell. The neighbourhood of the generic cell of coordinate $(i, j)$ is given by

$$\begin{aligned} V(X, (i,j)) = \\ = \{(i,j)+(0,0), (i,j)+(0, -1), (i,j)+(1, 0), \\ (i,j)+(-1, 0), (i,j)+(0, 1)\} = \\ = \{(i,j), (i,j\text{-}1), (i\text{+}1,j), (i\text{-}1,j), (i,j\text{+}1)\} \end{aligned} \quad (1)$$

- $Q$ is the set of cell states; it is subdivided in the following substates:
  - $Q_z$ is the set of values representing the topographic altitude (m);
  - $Q_h$ is the set of values representing the lava thickness (m);
  - $Q_T$ is the set of values representing the lava temperature (K°);
  - $Q_o^5$ are the sets of values representing the lava outflows from the central cell to the neighbouring ones (m).

  The Cartesian product of the substates defines the overall set of state $Q$:

$$Q = Q_z \times Q_h \times Q_T \times Q_o^5$$

- $P$ is set of global parameters ruling the CA dynamics:
  - $P_T = \{T_{vent}, T_{sol}, T_{int}\}$, the subset of parameters ruling lava viscosity, which specify the temperature of lava at the vents, at solidification and the "intermediate" temperature (needed for computing lava adherence), respectively;
  - $Pa = \{a_{vent}, a_{sol}, a_{int}\}$, the subset of parameters which specify the values of adherence of lava at the vents, at solidification and at the "intermediate" temperature, respectively;
  - $p_c$, the cooling parameter, ruling the temperature drop due to irradiation;
  - $p_r$, the relaxation rate parameter, which affects the size of outflows.

- $\sigma : Q^5 \rightarrow Q$ is the deterministic cell transition function. It is composed by four "elementary processes", briefly described in the following:

  σ1. *Outflows computation*. It determines the outflows from the central cell to the neighbouring ones by applying the minimisation algorithm of the differences; note that the amount of lava which cannot leave the cell, due to the effect of viscosity, is previously computed in terms of adherence. Parameters involved in this elementary process are: $P_T$ and $P_a$.

  σ2. *Lava thickness computation*. It determines the value of lava thickness by considering the mass exchange among the cells. No parameters are involved in this elementary process.

  σ3. *Temperature computation*. It determines the lava temperature by considering the

temperatures of incoming flows and the effect of thermal energy loss due to surface irradiation. The only parameter involved in this elementary process is $p_c$.

σ4. *Solidification*. It determines the lava solidification when temperature drops below a given threshold, defined by the parameter $T_{sol}$.

### 2.2. The MCA debris flow model SCIDDICA

SCIDDICA is a family of bi-dimensional MCA debris flow models, successfully applied to the simulation of many real cases, such as the 1988 Mt. Ontake (Japan) landslide (Di Gregorio et al. 1999), and the 1998 Sarno (Campania, Italy) disaster (D'Ambrosio et al. 2003). In this work, the basic version "T" of SCIDDICA (Avolio et al. 2000) was considered and its application to the 1992 Tessina (Italy) landslide shown. For a more in-depth knowledge of debris flow modelling using MCA approach, please refer to (D'Ambrosio et al. 2003)

SCIDDICA considers the surface over which the phenomenon evolves as subdivided in square cells of uniform size. Each cell changes its state by means of the transition function, which takes as input the state of the cells belonging to the von Neumann neighbourhood. It is formally defined as

$$SCIDDICA = < R, X, Q , P, \sigma >$$

where:

- $R$ is the set of points, with integer coordinates, which defines the 2-dimensional cellular space over which the phenomenon evolves. The generic cell in $R$ is individuated by means of a couple of integer coordinates $(i, j)$, where $0 \leq i < i_{max}$ and $0 \leq j < j_{max}$.
- $X = \{(0,0), (0, -1), (1, 0), (-1, 0), (0, 1)\}$ is the von Neumann neighbourhood relation, a geometrical pattern which identifies the cells influencing the state transition of the central cell. The neighbourhood of the generic cell of coordinate $(i, j)$ is given by

$$V(X, (i,j)) = \\ = \{(i,j)+(0,0), (i,j)+(0, -1), (i,j)+(1, 0), \\ (i,j)+(-1, 0), (i,j)+(0, 1)\} = \\ = \{(i,j), (i,j\text{-}1), (i\text{+}1,j), (i\text{-}1,j), (i,j\text{+}1)\} \quad (2)$$

- $Q$ is the set of cell states; it is subdivided in the following substates:
  - $Q_z$ is the set of values representing the topographic altitude (i.e. elevation);
  - $Q_h$ is the set of values representing the debris thickness;
  - $Q_o^5$ are the sets of values representing the debris outflows from the central cell to the neighbouring ones (recall that the central cell is part of its neighbourhood).

  The Cartesian product of the substates defines the overall set of state Q:

$$Q = Q_z \times Q_h \times Q_o^5$$

- $P$ is set of global parameters ruling the CA dynamics:
  - $p_a$ is the parameter which specifies the thickness of the debris that cannot leave the cell due to the effect of adherence;
  - $p_z$ is the critical altitude, defining two regions of different rheological behaviours of the flow; it is related to parameters $p_f$;
  - $p_f$ is the "friction angle" parameter, which empirically defines the minimum slope between two cells needed for debris motion - its value is related to that of the parameter $p_z$;
  - $p_r$ is the relaxation rate parameter, which affects the size of outflows (cf. section above).

- $\sigma : Q^5 \rightarrow Q$ is the deterministic cell transition function. It is composed by two "elementary processes":
  - $\sigma_1 : (Q_z \times Q_h)^5 \times p_a \times p_z \times p_f \times p_r \rightarrow Q_o^5$ determines the outflows from the central cell to the neighbouring ones by applying the minimisation algorithm of the differences. In brief, a preliminary control eliminates those cells for which the slope with respect the central cell is less than $p_f$. Moreover, the amount of debris which cannot leave the cell, due to the effect of viscosity, is simply modelled by means of the parameter $p_a$. Thus, by means of the minimization algorithm, outflows $q_o(0,i)$ ($i$=0,1,…,4) are evaluated and the substates $Q_o^5$ accordingly updated. Note that $q_o(0, 0)$ represents the amount of debris that does not flow out of the central cell. Eventually, a relaxation rate factor, $p_r \in ]0,1]$, can be considered in order to obtain the local equilibrium condition in more than one CA step. This can significantly improve the realism of model as, in general, more than one step may be needed to displace the proper amount of debris from a cell towards the adjacent ones. In this case, if $f(0,i)$ ($i$=1, …, 4) represent the outgoing flows towards the 4 adjacent cells, the resulting outflows are given by $q_o(0,i)=f(0,i)\cdot p_r$ ($i$=1, …, 4), while the amount of debris remaining in the central cell is obtained as:

$$h_r(0) = q_0(0,0) = h(0) - \sum_{i=1}^{4} q_0(0,i) \quad (3)$$

- $\sigma_2 : (Q_o^5)^5 \rightarrow Q_h$ determines the value of debris thickness inside a cell by considering mass exchange in the cell neighbourhood: $h(0) = \sum_i (q_o(0,i) - q_o(i,0))$, where $i = 0,1,\ldots,4$ and $q_o(i,0)$ represents the inflow from the $i^{th}$ adjacent cell. The substate $Q_h$ is accordingly updated. Note that no

parameters are involved in this elementary process.

## 3. THE LIBAUTOTI LIBRARY FOR MACROSCOPIC CELLULAR AUTOMATA

The MCA approach permits to straightforwardly define a simulation model of a complex system, such as a lava flow. Moreover, the MCA models of complex systems often need to be the most efficient possible since, depending also on the size of input data, each simulation can last days or even weeks (cf. D'Ambrosio et al. 2006). As a consequence, the developer must implement proper optimization strategies (e.g. cf. Walter and Worsch 2004) and, when mandatory, parallelize the program (e.g. by means of MPI – Message Passing Interface).

Unfortunately, the world of development tools for MCA suffers of a lack of open-source software and often the developer must pay attention to all the low levels details of the implementation, such as memory allocation/de-allocation, I/O management and so on.

To overcome these difficulties, the developer of a MCA model could decide to uneffortlessly implement her/his own simulation model by scratch using a high level language (such as C++ or FORTRAN) or consider a valid (but not-free) software solution, namely the CAMELot CA simulation environment. Advantages of the last solution include the use of a proprietary language (i.e. the CARPET one, cf. Spezzano and Talia 1998) for model definition, integrated 2D/3D viewer and the possibility to run the simulation in parallel by means of the Message Passing paradigm, in a completely transparent manner to the user (i.e. she/he does not need to care of parallel issues such as process allocation on nodes, data partitioning, global-reduction operations, etc). However, disadvantages include the not extensibility of the library, lack of adequate debug facilities and reduced 3D visualization capabilities.

As consequence of all these last considerations, the *libAuToti* library can represent a valid solution. Differently to CAMELot, *libAuToti* is not an integrated simulation environment, but an ANSI C++ (thus portable) library, which intends to offer the main features of CAMELot, as the possibility to simply define the model by ignoring low level details, manage input and output data and execute the simulation both sequentially or in parallel by adopting the Message Passing paradigm.

Differently to CAMELot, the developer produces a standard C++ program by including the *libAuToti* library header file, with all the advantages that this approach permits. Among these, the possibility to choose the preferred development environment with the related debug facilities (not very functional in CAMELot), and the possibility to easily introduce further features such as an optimization module or a 3D viewer. In the following examples, CA model development is illustrated by considering the *libAuToti* library on the SCIARA lava flows and SCIDDICA debris flows models. Furthermore, computational results of tests carried out on an Alpha Server SC45 supercomputer and on an Intel dual-core PC are illustrated and commented.

### 3.1. A libAuToti implementation of SCIARA

As stated above, the library has been tested by considering the Macroscopic Cellular Automata model SCIARA for lava flow simulation. In the following, a seamless parallel implementation of SCIARA is presented. In particular, Figure 1 and 2 illustrate the CA definition and transition function, respectively. Besides common C++ libraries such as `iostream`, `math` or `vector` (here omitted for briefness), the program includes the `libAuToti.h` header that allows to employ the library. In particular, the object `sciara` is defined, representing a MCA with the possibility to perform a parallel simulation. As it can be easily seen, the user does not need to concentrate in parallel issues, such as the CA space decomposition or message passing of boundary "ghost" cells between nodes: all these operations are transparently carried out by the library. Moreover, once the program has been compiled, the user has to simply decide how many processing elements the CA has to run on (e.g. `mpirun –np 4 sciara` for automatically executing sciara in parallel on 4 processing nodes). Note that, in order to perform a sequential simulation, it will be sufficient to define an object of type `SeqSimulationCA_2D`.

```
// header files here...
#include "libAuToti.h"

int main(int argc, char** argv)
{
   ParallelSimulationCA_2D sciara;
   sciara.setDim(410,296);

   //flow substates
   sciara.addSubstate(SUBSTATE_FLOAT);// 0
   sciara.addSubstate(SUBSTATE_FLOAT);// 1
   sciara.addSubstate(SUBSTATE_FLOAT);// 2
   sciara.addSubstate(SUBSTATE_FLOAT);// 3
   //elevation substate
   sciara.addSubstate(SUBSTATE_FLOAT );//4
   //lava thickness substate
   sciara.addSubstate(SUBSTATE_FLOAT); //5
//lava temeperature substate
   sciara.addSubstate(SUBSTATE_FLOAT); //6


   sciara.setRadius( 1 );

   // von Neumann neighbourhood
   sciara.addNeighboor( 0,-1 );
   sciara.addNeighboor( 1, 0 );
   sciara.addNeighboor( -1, 0 );
   sciara.addNeighboor( 0, 1 );

   sciara.addParameter(1373); //vent temp.
   sciara.addParameter(1153); //solidif. temp.
   sciara.addParameter(1265); //interm. temp.
   sciara.addParameter(0.7); //vent adherence
   sciara.addParameter(7); //solidif. adher.
   sciara.addParameter(1.5); //interm. adher.
   sciara.addParameter(1.4e-14); //cool. par.
   sciara.addParameter(0.6); //relaxation

   sciara.initialize(argc, argv);
```

```
    sciara.readFromFile( "./nicolosi.txt", 4);
    sciara.readFromFile( "./vent_pos.txt", 5);

    sciara.setNumStep( 18000 );

    sciara.run();

    sciara.finalizeSimulation();
    return 0;
}
```

Figure 1: The SCIARA CA definition by using the *libAuToti* free MCA library.

The member function `setDim` defines the cell space dimensions, while `setRadius` and `addNeighboor` the CA the neighbourhood radius and relation, respectively.

Table 1: List of all allowed type for the substates definition together with their corresponding C++ types.

| *libAuToti* substate type | Corresponding C++ type |
|---|---|
| SUBSTATE_CHAR | char |
| SUBSTATE_SHORT | short |
| SUBSTATE_INT | int |
| SUBSTATE_LONG | long |
| SUBSTATE_FLOAT | float |
| SUBSTATE_DOUBLE | double |

As required by SCIARA seven substates are defined thanks to the `addSubstate` member function. Table 1 lists all the allowed substates type and their identifiers. Once a substate is defined, a numerical handle is automatically defined in order to refer the substate in the program. However, to simply refer to a substate, the user can declare an enumerative type; for the case of SCIARA it could be the following:

```
enum Subs{
    FLOW0 = 0,  //outflow Qo towards the N cell
    FLOW1,   //outflow Qo towards the E cell
    FLOW2,   //outflow Qo towards the W cell
    FLOW3,   //outflow Qo towards the S cell
    ELEV,    //cell elevation a.s.l Qz
    TEMP,    //cell temperature (K°)
    THICK    //lava thickness Qh inside cell
};
```

In this way, it is possible to refer to a substate through a symbolic identifier, instead of a numerical one.

As requested by the considered version of SCIARA, the von Neumann neighbourhood was defined, thanks to the `addNeighboor` member function. Note that, similarly for the case of substates, a numerical handle is implicitly assigned to each cell of the neighbourhood, except for the central cell that the library directly refers (cf. below). The library also permits the definition of more "sophisticated" neighbourhoods, as Moore's one (both square or hexagonal) by the way required by the latest releases of SCIDDICA (D'Ambrosio et al. 2003, Iovine et al. 2005).

Parameters are defined by means of the `addParameter` member function. As before, a numerical handle is automatically defined by the library, and the user can define an enumerative type to better manage parameters setting and usage; it could be the following (note that the first 8 correspond to SCIARA parameters shown in Section 2.1):

```
enum Params{
    TEMP_V = 0,    //temp. at vent
    TEMP_S,        //temp. at solidification
    TEMP_I,        //interm. temp
    ADHERENCE_V,   //adherence at vent
    ADHERENCE_S,   //adherence at solid.
    ADHERENCE_I,   //interm. adherence
    COOL,          //cooling parameter
    RELAXATION,    //relaxation
    IS_VENT,       //cell is a crater?
    VENT_THICK     //crater lava thickness
};
```

Parameter setting and retrieval are managed by the two self-explanatory member functions `setParameter` and `getParameter`, respectively. Moreover, input data is automatically performed by the function `readFromFile`, which reads files in standard ASCII Grid format.

Eventually, the simulation is performed (for 18000 steps) by means of the function `run()`.

Figure 2 shows the *libAuToti* implementation of the SCIARA transition function, defined as an inline member function. Recall that, by CA definition, this function is called at *each* step. Substates identifiers, together with the neighbours ones, are utilised as arguments of `getSubstate*`, which returns the value of a certain substate for the specified neighbouring cell. Here the symbol `*` can be one of the types listed in Table 1.

```
inline      void      parallelSimulationAC2D      ::
transitionFunction()
{
// set lava width for crater
if(getSubstateFloat(ELEV)==getParameter(IS_CRATER))
   updateCellSubstate(THICK,
            getParameter(vent_thick));

switch (STEP%2){
    case 1:  calc_flows(this);
         break;
    case 0:  calc_thickness(this);
         calc_temperature(this);
         calc_quote(this);
         break;
    }
}
```

Figure 2: The SCIARA transition function definition by using the *libAuToti* free MCA library.

For instance, the following portion of code verifies if the value of the substate `ELEV` of type `float`, is equal to the `IS_CRATER` parameter, meaning that the cell itself represents a crater:

```
if(getSubstateFloat(ELEV)==getParameter(IS_CRAT
ER))
```

Analogously, if one wants to check the value of a neighbouring cell, she/he has to simply add the neighbour index as second argument to the

`getSubstateFloat` function. The following example could test if the value of the substate `THICK` of the neighbour number 2 is equal to 0:

```
if (getSubstateFloat( THICK, 2 ) == (float)0 )
```

Equivalently, the function `updateCellSubstate` updates the value of a given substate for the central cell. For example, the following portion of code sets the new value for the substate `THICK` for the central cell to the value of the `vent_thick` parameter (i.e. the amount of lava that is discharged by the crater at each CA step).

```
updateCellSubstate(THICK,
    getParameter(vent_thick));
```

Besides the setting of the value of lava thickness for the crater, the transition function subsequently calls alternatingly the `calc_flow` function and the `calc_thickness`, `calc_temperature` and `calc_quote` functions (here all omitted for brevity), depending on the fact the CA step, whose value is given by the internal variable `STEP`, is odd or even. All four functions correspond to the elementary processes $\sigma_1$, $\sigma_2$, $\sigma_3$ and $\sigma_4$, respectively (cf. Section 2.1). In this way, a single SCIARA step corresponds to 2 executions of the *libAuToti* transition function (double-step transition function).

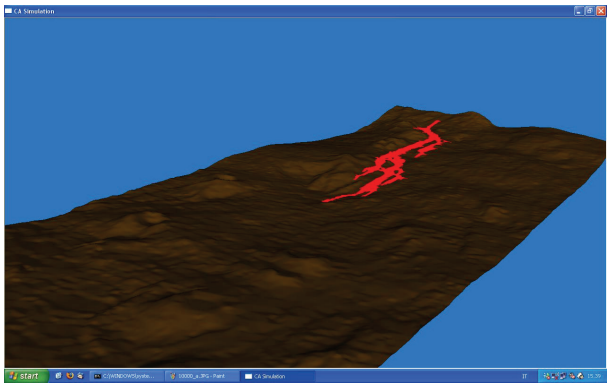

Figure 3: The SCIARA MCA application to the simulation of the 2001 Mt Etna event at Nicolosi. The figure, shows a snapshot of the CA configuration of the system after 18000 steps with the lava path in red, utilizing the libAuToti OpenScenegraph 3D viewer.

As stated above, though an integrated environment surely ensures many advantages, a benefit of using a separate library is the more flexibility which, for instance, allows the developer to easily extend the software features with the aid of further libraries. This is the case of the SCIARA and SCIDDICA implementations, where the computational model was written by utilizing the *libAuToti* library and two 3D visualization modules developed with OpenGL and with the OpenScenegraph visual toolkit. Figure 3 shows a snapshot of the SCIARA model simulation on the Nicolosi 2001 lava event at the end of the simulation with the latter viewer.

At the same way, it would be possible add a further module, e.g. for model optimization purposes, by using the PGAPack (PGApack web reference 1998) free library for Parallel Genetic Algorithms (PGAs) (cf. also Iovine et al. 2005 for an application of PGAs for optimizing parameters of a MCA model).

### 3.2. A libAuToti implementation of SCIDDICA

In this paragraph, a seamless parallel implementation of SCIDDICA is presented. In particular, Figure 4 and 5 illustrate the CA definition and transition function, respectively. As in the SCIARA libAuToti implementation, the program includes the `libautoti.h` header that allows to employ the library. In particular, the object `sciddica` of type `ParallelSimulationCA_2D` is defined, representing a MCA with the possibility to perform a parallel simulation.

```
// header files here...
#include "libAuToti.h"

int main(int argc, char** argv)
{
   ParallelSimulationCA_2D sciddica;
   sciddica.setDim(410,296);

   //flow substates
   sciddica.addSubstate(SUBSTATE_FLOAT);// 0
   sciddica.addSubstate(SUBSTATE_FLOAT);// 1
   sciddica.addSubstate(SUBSTATE_FLOAT);// 2
   sciddica.addSubstate(SUBSTATE_FLOAT);// 3
   //elevation substate
   sciddica.addSubstate(SUBSTATE_FLOAT );//4
   //width substate
   sciddica.addSubstate(SUBSTATE_FLOAT); //5

   sciddica.setRadius( 1 );

   // von Neumann neighbourhood
   sciddica.addNeighboor( 0,-1 );
   sciddica.addNeighboor( 1, 0 );
   sciddica.addNeighboor(-1, 0 );
   sciddica.addNeighboor( 0, 1 );

   sciddica.addParameter(0.5); //adherence
   sciddica.addParameter(900); //crit. elevat.
   sciddica.addParameter(0.3); //fric. param.
   sciddica.addParameter(0.6); //relaxation

   sciddica.initialize(argc, argv);
   sciddica.readFromFile( "./tessina.txt", 4);
   sciddica.redFromFile( "./source.txt", 5);

   sciddica.setNumStep( 10000 );

   sciddica.run();

   sciddica.finalizeSimulation();
   return 0;
}
```

Figure 4: The SCIDDICA CA definition by using the libAuToti free MCA library.

The member function `setDim` defines the cell space dimensions, while `setRadius` and `addNeighboor` the CA the neighbourhood radius and relation, respectively. Moreover, as required by SCIDDICA, six substates are defined thanks to the `addSubstate` member function.

As requested by the considered version of SCIDDICA, the von Neumann neighborhood was defined, thanks to the `addNeighboor` member function.

As in the SCIARA libAuToti implementation, parameters are defined by means of the `addParameter` member function.

Moreover, input data is automatically performed by the function `readFromFile`, which reads files in standard ASCII Grid format. Eventually, the simulation is performed (for 10000 steps) and results saved (even in ASCII Grid format) by means of the function `run()`.

Figure 5 shows the libAuToti implementation of the SCIDDICA transition function, defined as an inline member function. Substates identifiers, together with the neighbours ones, are utilised as arguments of `getSubstate*`, which returns the value of a certain substate for the specified neighbouring cell. Here the symbol * can be one of the types listed in Table 1.

```
inline void parSimulationAC2D::transitionFunction()
{
if(getSubstateFloat(ELEV)>getParameter(DETACHMENT))
    setParameter(FRICTION,0.3);
else
    setParameter(FRICTION,0.5);

if (STEP == 0)
    if (getSubstateFloat(WIDTH) > (float)0)
            updateCellSubstate(ELEV,
                               (getSubstateFloat(ELEV)-
                                getSubstateFloat(WIDTH))
                               );
switch (STEP%2){
    case 1:  calc_flows(this);
                break;
    case 0:  calc_width(this);
                break;
    }
}
```

Figure 5: The SCIDDICA transition function definition by using the libAuToti free MCA library.

For instance, the following portion of code verifies if the value of the substate `WIDTH` of type `float`, is greater than 0 for the central cell:

```
if ( getSubstateFloat( WIDTH ) > (float)0 )
```

Analogously, if one wants to check the value of a neighbouring cell, she/he has to simply add the neighbour index as second argument to the `getSubstateFloat` function. The following example tests if the value of the substate `WIDTH` of the neighbour number 2 is equal to 0:

```
if (getSubstateFloat( WIDTH, 2 ) == (float)0 )
```

Equivalently, the function `updateCellSubstate` updates the value of a given substate for the central cell. For example, the following portion of code sets the new value for the substate WIDTH for the central cell to the value `new_width`.

```
updateCellSubstate(WIDTH, new_width );
```

Besides the setting of the $p_f$ parameter (cf. FRICTION in Figure 5), the transition function performs a preliminary elaboration at step 0 in order to properly handle input data. Subsequently, it calls one of the two functions `calc_flow` or `calc_width` (here omitted for brevity), depending on the fact the CA step, whose value is given by the internal variable `STEP`, is odd or even. They correspond to the elementary processes $\sigma_1$ and $\sigma_2$, respectively (cf. Section 2.2). In this way, a single SCIDDICA step corresponds to 2 execution of the libAuToti transition function (double-step transition function). Accordingly, the SCIDDICA simulation steps were indeed exactly 5000.

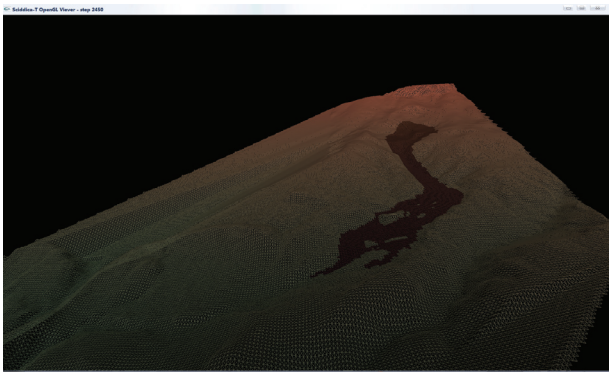

Figure 6: The SCIDDICA MCA application to the simulation of the Tessina Landslide. The figure shows a snapshot of the CA configuration of the system after 5000 steps utilizing the libAuToti OpenGL 3D viewer

### 3.3. Computational results

As concerns computational results, the SCIARA and SCIDDICA models – as implemented by adopting the *libAuToti* library – were tested on two parallel computers: a 24 processor HP Alpha Server SC45 and an Intel T2500 2Ghz dual-core PC.

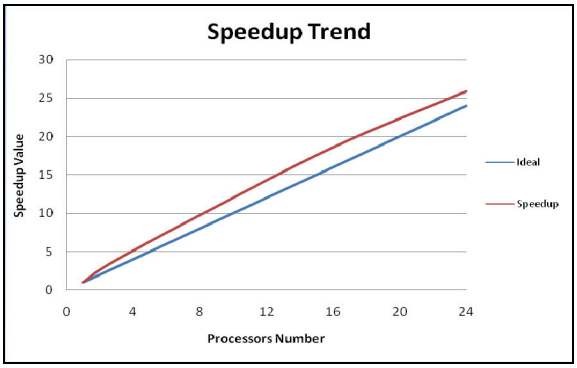


Figure 7: Speed-up of *libAuToti* for the model SCIARA carried out on a 24 processor Alpha Server SC45. Superlinear speedup is evident up to 24 processors.

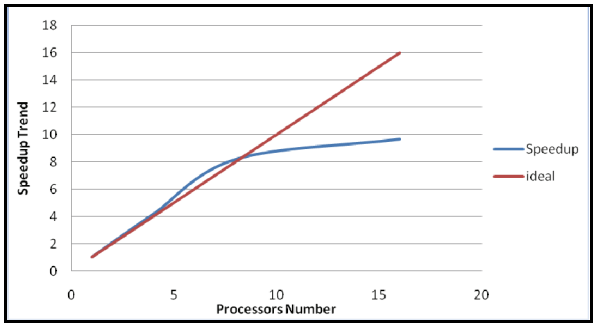


Figure 8: Speed-up of libAuToti for the MCA model SCIDDICA. Results refer to experiments carried out on 16 processors of the Alpha Server SC45. Superlinear speedup is evident up to 8 processors.

In particular, the high performing computer is composed of 6 SMP nodes, each with 4 Alpha 64-bit 1250 Mhz EV68 processors and 16 MB cache. Figure 7 shows the speed-up diagram for experiments carried out on the high performing machine, referred to the SCIARA model. In this case, super-linear speed-up was achieved up to 24 processors, probably due to the limited dimensions of the CA space and cache memory effects. Figure 8 depicts the speed-up diagram, referred to the SCIDDICA model experiments performed adopting 16 processors. Even in this case results, though slightly worse than the previous experiments, show the good scalability of the parallel system, together with super-linear effects up to 8 processors. Eventually, experiments carried out on the low cost machine – the Intel T2500 2Ghz dual-core PC, here omitted for briefness – have been positive. In particular, with respect with the first case, results have shown an even better super-linear effect when executed on both cores. Still, this is probably due both to the limited dimensions of the CA space and for the greater cache-effect (i.e. cache is physically closer to the CPU in multi-core machines) of the dual core architecture with respect to the supercomputer.

However, in all cases, computational results evidenced the goodness of the library in exploiting the considered parallel systems, confirming that even low-cost off-the-shelf machines can be fruitfully employed.

## 4. CONCLUSIONS

We have presented examples of implementation of the *libAuToti* C++ parallel library for Scientific Computing of two models based on the Macroscopic Cellular Automata (MCA) approach. In particular, the two considered models regard SCIARA, for lava flow simulation, and SCIDDICA, for debris flow modeling. The library, particularly suitable for MCA modeling, is easily extensible and easily combinable with different libraries, such as optimization or visualization modules.

Moreover, computational performances, achieved on a 24 processor Alpha Server SC45 supercomputer and on an Intel dual-core processor based machine have demonstrated the goodness of the library. Future developments of the library will regard the implementation of a load balance strategy (e.g. based on the scattered decomposition of the computational domain), further library core optimization and the integration of an interactive console (e.g. similar to that of GNUPlot or Octave) to better control the phases of the simulation. Eventually, a full integration with interactive OpenGL or OpenScenegraph scientific 3D viewers will be tackled in the future. The library is freely downloadable from its official website, at the URL http://autoti.mat.unical.it. The library's name, *libAuToti*, is gaily taken from Prof. Salvatore "Toti" Di Gregorio, inventor of the MCA approach.

## ACKNOWLEDGMENTS

A special thanks goes to Prof. Gino "Big Chief" Mirocle Crisci and Dr. Valeria Lupiano for the common researches.

## REFERENCES

Avolio, M.V., Di Gregorio, S., Mantovani, F., Pasuto, A., Rongo, R., Silvano, S., Spataro, W., 2000. Simulation of the 1992 Tessina landslide by a cellular automata model and future hazard scenarios, *International Journal of Applied Earth Observation and Geoinformation* 2:41-50.

Barca, D., Crisci, G.M., Di Gregorio, S., Nicoletta, F., 1994. Cellular Automata for simulating lava Flows: A method and examples of the Etnean eruptions. *Transport Theory and Statistical Physics* 23:195-232.

Crisci, G.M., Rongo, R., Di Gregorio, S., Spataro, W., 2004. The simulation model SCIARA: the 1991 and 2001 lava flows at Mount Etna. *Journal of Volcanology and Geothermal Research*, 132:253-267.

D'Ambrosio D., Di Gregorio, S., Iovine, G., Lupiano, V., Rongo, R., Spataro, W., 2003. First simulations of the Sarno debris flows through Cellular Automata modeling. *Geomorphology*, 54:91-117.

D'Ambrosio, D., Rongo, R., Spataro, W., Avolio, M.V., Lupiano, V., 2006. Lava Invasion Susceptibility Hazard Mapping Through Cellular Automata. *Proceedings of ACRI 2006*, pp. 452-461. September 20-23, Perpignan, France.

D'Ambrosio, D.; W. Spataro; and G. Iovine. 2006. Parallel genetic algorithms for optimising cellular automata models of natural complex phenomena: an application to debris-flows. *Computers and Geosciences-UK*, 32, 861-875.

Di Gregorio, S., Festa, D.C., Rongo, R., Spataro, W., Spezzano, G., Talia, D., 1996. A microscopic freeway traffic simulator on a highly parallel system. In: D'Hollander, E.H., Joubert, G.R., Peters D. Trystam, F.J., eds. *Parallel Computing: State-of-the-Art and Perspectives*, Amsterdam:Springer, 69-76.

Di Gregorio, S., Mele, F., Minei, G., 2001. Automi Cellulari Cognitivi. Simulazione di evacuazione, Proceedings of Input - Conferenza Nazionale

Informatica Pianificazione Urbana e Territoriale, Democrazia e Tecnologia, cdrom, June 23-26, Isole Tremiti, Italy.

Di Gregorio, S., Rongo, R., Siciliano, C., Sorriso-Valvo, M., Spataro, W, 1999. Mount Ontake landslide simulation by the cellular automata model SCIDDICA-3, *Physics and Chemistry of the Earth Part A* 24:97-100.

Di Gregorio, S., Serra, R., 1999. An empirical method for modelling and simulating some complex macroscopic phenomena by cellular automata. *Future Generation Computer Systems* 16:259-271.

Iovine, G., D'Ambrosio, D., Di Gregorio, S., 2005. Applying genetic algorithms for calibrating a hexagonal cellular automata model for the simulation of debris flows characterised by strong inertial effects. *Geomorphology* 66:287-303.

McBirney, A.R., Murase, T., 1984. Rheological properties of magmas. *Annual Review of Earth Planetary Sciences*, 12:337-357.

PGAPack web reference: www-fp.mcs.anl.gov/CCST/research/reports_pre1998/comp_bio/stalk/pgapack.html.

Spataro, W., D'Ambrosio, D., Spingola, S., Zito, G., Rongo, R., 2008. LibAuToti, A Parallel Cellular Automata Library for Simulation: An example of Application to Landslides. *Proceedings of Summer Computer Simulation Conference*, June 16-19, Edinburgh, UK).

Spezzano, G., Talia, D., 1998. Designing Parallel Models of Soil Contamination by the CARPET Language, *Future Generation Computer Systems* 13:291-302.

Succi, S., 2004. *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*. London, Oxford:University Press.

Trunfio, G.A., 2004. Predicting Wildfire Spreading Through a Hexagonal Cellular Automata Model, *Proceedings of ACRI 2004*, pp. 725-734. October 23-25, Amsterdam, The Netherlands.

Walter, R., Worsch, T., 2004. Efficient Simulation of CA with Few Activities. *Proceedings of ACRI 2004*, pp. 101-110. October 23-25, Amsterdam, The Netherlands.

## AUTHORS BIOGRAPHY

**WILLIAM SPATARO** was born in London, UK. He is an Assistant Professor in Computer Science at the Department of Mathematics at the University of Calabria (Italy). His principal interests regard Parallel Computing, Cellular Automata, Modelling and Simulation of complex phenomena and Genetic Algorithms. His e-mail address is: spataro@unical.it and his Web-page can be found at http://www.mat.unical.it/spataro.

**DONATO D'AMBROSIO** was born in Cosenza, Italy. He is an Assistant Professor in Computer Science at the Department of Mathematics at the University of Calabria (Italy). His interests concern Genetic Algorithms, Parallel Computing, Simulation of Complex Natural Phenomena and Computer Graphics. His e-mail address is: d.dambrosio@unical.it and his Web-page can be found at http://www.mat.unical.it/~donato.

**ROCCO RONGO** was born in Naples, Italy. He is an Assistant Professor in Computer Science at the Department of Earth Sciences at the University of Calabria (Italy). His major interests regard Parallel Computing, Modelling and Simulation of Geological processes by Cellular Automata and GIS software. His e-mail address is: rongo@unical.it and his Web-page can be found at http://dister.unical.it/rongo/.

**GIUSEPPE SPINGOLA** was born in Praia a Mare, in Calabria (Italy). His Computer Science Degree thesis regarded the first release of the libAutoti library. He presently is working with the Department of Mathematics and the Department of Chemical Engineering of the University of Calabria for code parallelization of simulation models and on further releases of the libAuToti library. His email address is g.spingola@gmail.com.

**GIUSEPPE ZITO** was born in Torino, Italy. His Computer Science Degree thesis regarded the first release of the libAutoti library. He currently is collaborating with the Department of Mathematics and the Department of Chemical Engineering of the University of Calabria for code parallelization of simulation models on further releases of the libAuToti library. His email address is giuseppe.zito@gmail.com.

**MARIA VITTORIA AVOLIO** was born in Cosenza, Italy. Mathematician and Researcher Assistant at the Department of Mathematics and at the University of Calabria, Italy, she is involved basically in Modelling and Simulation of lava, landslide and pyroclastic flows by Cellular Automata methods. She is the beloved wife of William Spataro, who fell in love with her during her PhD course in Computer Science and Mathematics at the same University. Her e-mail address is: avoliomv@unical.it.

**SALVATORE DI GREGORIO** was born in Castellamare del Golfo, Sicily, Italy. He is Full Professor in Computer Science at the Department of Mathematics at the University of Calabria (Italy). Together with Professor Gino Mirocle Crisci, he leads the Empedocles Research group at the same University. His research interests concern Computational Models of Complex Systems and Parallel Computing. His e-mail address is: dig@unical.it.