Designing Parallel Models of Soil Contamination by the CARPET Language

Giandomenico Spezzano and Domenico Talia

ISI-CNR, c/o DEIS, Università della Calabria, 87036 Rende (CS), Italy

Phone: +39 984 839047, Fax: +39 984 839054

Email: `{spezzano, talia}@si.deis.unical.it`

Abstract

This paper describes the main features of the CARPET language and its practical use for programming three-dimensional models of the contamination of soils developed in the CABOTO project. CARPET is a high-level language based on the cellular automata model, which supports rapid prototyping of a large number of applications in science and engineering. A CARPET implementation has been used for programming cellular algorithms in the CAMEL parallel system. The CAMEL (Cellular Automata environMent for systEms modeLing) system is a parallel implementation of a software environment for the simulation and modeling of complex systems based on cellular automata. CAMEL offers the computing power of a parallel computer although hiding, by the CARPET language, the architecture issues to a user. The CARPET language allows the design of parallel programs for describing the actions of thousands of simple active interacting agents that might simulate the behavior of very complex systems.

Keywords: Parallel programming; Cellular automata; High performance computing; Complex system modeling;

1. Introduction

Computational science is a new research area that it is based on the use of computers for solving scientific problems. In recent years, computer simulation and modeling is increasingly becoming a very useful tool to assist research and experiments in science and engineering. In fact, computer-based simulations are used in science in some cases as a valid alternative and in other cases for additional verification of laboratory experiments. In engineering, simulation can be used to support system design since it allows prediction of the performance of a product based on data design only, without the need to build a real prototype [18].

The high computing power offered by available parallel computers determines the feasibility of the computational science approach for solving real world problems. Massively parallel computers allow simulating scientific and engineering problems exploiting high performance. They represent the most effective computing support for modeling and simulation of large complex phenomena and systems. Parallel computers are used to support the classical approach based on differential equations. Furthermore, they are well suited for implementing new inherently parallel computing abstract models such as cellular automata (CA) [16], neural networks, and genetic algorithms that represent significant mathematical models for describing complex systems in science and engineering.

In particular, CAs are dynamical systems in which space and time are discrete [17]. A cellular automaton consists of a regular grid of cells, each of which can be in one of a finite number of n possible states, updated synchronously (in parallel) in discrete time steps according to a local identical interaction rule [14]. The state of a cell is determined by the previous states of a surrounding neighborhood of cells. CA are an intrinsic machine-independent parallel model of computation that can be used to efficiently implement programming environments that allow a easier access to parallel machine facilities hiding many of the complexities of the underlying parallel architecture. Furthermore, these software environments allow a user to do rapid prototyping and steering of a large number of applications in science and engineering by simple and easy-to-use graphical user interfaces (GUI) and a high level language. Following this idea we have built a parallel CA programming environment called CAMEL and a new high level language called CARPET to allow users with a minimum computational background in parallel computing to design and visualize simulations of complex phenomena. This paper discusses, as a significant example of this approach, the modeling and simulation of soil contamination by using a cellular-automata based design.

Soil decontamination is a major environmental problem in industrial and third-world countries: while different decontamination methods are known, the evaluation of their effectiveness requires a number of laboratory or pilot plant tests. However, appropriate forecasting of real field results from these tests, in order to reliably evaluate the times and costs of intervention, could be achieved only by resorting to mathematical models, which are fairly complicated and require high computing power. An important goal in order to face soil pollution problems is the ability to simulate contaminant transport, diffusion and transformation processes for reliably estimating their impact and forecasting the effects of different decontamination strategies.

Here we propose a three dimensional cellular automata model, where the fluid flow with diffusion-transport [2,8] of contamination agents inside the soil and their mutual influences are viewed as a dynamic system based on local interactions with discrete time and space, where the space is represented by cubic cells. A cell is characterized by specific values (the state) of selected physical quantities, representing physical-chemical specifications relevant to the evolution of the phenomenon. States of all cells in the automaton are updated by means of an identical local rule called *transition function*, which determines the cell's next state upon the states of its neighbors. The evolution of the states of all cells determines the global behavior of the system as a result of multiple interactions.

In this paper, we intend to show how the CARPET language and the CAMEL environment can be used to implement the simulation of the contamination of soils developed in the CABOTO project funded by the PCI ESPRIT programme. Our approach here is to devote enough space for discussing algorithms details and programming issues that generally are not addressed and discussed in the majority of papers on CA modeling and simulation. According to this approach, the rest of the paper is organized as follows. Section 2 gives a very short description of CAMEL.

Section 3 introduces the main features of the CARPET language and section 4 discusses in detail the design and programming of the soil contamination model by CARPET. Finally, section 5 presents figures and results in terms of accuracy and performance of the simulation model on a MIMD parallel architecture.

2. CAMEL

The main goal of CAMEL is to integrate computation, visualization, and control into one environment that allows interactive steering [15] of scientific applications on parallel computers [3]. CAMEL uses the CA theory both as a computational model of parallel processing and as a tool to simulate dynamical complex phenomena. On top of this architecture a high-level language has been designed that enables researchers to design models of discrete dynamical systems using the cellular automata formalism.

The CAMEL environment has been implemented on a message-passing MIMD parallel computer composed of a mesh of Transputers connected to a host node. It has three main components, the run-time system, the user interface and the visualization system. CAMEL implements a cellular automaton as a SPMD (Single Program, Multiple Data) program. Its parallel run-time system is composed of a set of *macrocell* processes each running on a single processing element of the parallel machine and a *controller* process running on a master processor. Each *macrocell* process implements several elementary cells and makes use of a communication system, which handles the data exchange among cells.

In CA, a stationary region is composed of cells with passive states that do not change state until the state of neighbor cells does not change too, and of cells with inert states that never change state. We implemented in CAMEL a load balancing strategy [1] that avoids computing the next state of cells that belong to a stationary region. Further, the load balancing strategy divides the computation of the next state of all the active cells among all the processors of the parallel machine.

In the design of the CAMEL architecture and run-time system, special attention has been paid to allow future porting on new parallel computers such as the Cray T3D, the Meiko CS2, the IBM SP2, and workstation clusters. In fact, CAMEL is going to be re-implemented using the C language plus the standard MPI library for achieving its portability on different multicomputer architectures.

3. The CARPET language

In [9] C. Pancake and D. Bergmark wrote that "Programming languages constitute the primary point of contact between scientist and machine" and "There is a sizeable gap between the user's conceptual solution to a problem and its ultimate realization on a parallel machine. Only narrowing that gap can we increase the reliability and effectiveness of scientific parallel programming." Other computational science researchers reaffirm these ideas. Many people experienced that parallel programming languages represent one of the main components in the development of computational science and engineering. In fact, programming languages influence at least three main aspects of the computational science process: algorithm design, solution implementation and program performance. For these reasons, it is crucial for engineers and scientists to have at one’s disposal high-level languages for designing computational science applications.

Cellular programming languages based on the cellular automata model represent a significant example of restricted-computation models [12] that are used to design parallel computation for a large number of applications in biology, physics, geophysics, chemistry, economics, artificial life, and engineering.

CARPET (*C*ellul*AR* *P*rogramming *E*nvironmen*T*) is a high-level language that extends C with some additional constructs to define data and describe the rules of the transition function of a single cell of a cellular automaton [13]. CARPET is a cellular programming language defined to design CA algorithms that implements a cellular automaton as a SPMD program. According to this approach, a user must specify only the transition function of a single cell of the system he wants to simulate by CARPET [4], then the transition function is executed in parallel to update the state of each cell.

The main features of CARPET are the possibility to describe the state of a cell as a set of typed substates each one by a user-defined type, and the simple definition of complex neighborhoods (e.g., hexagonal) that can be also time dependent in a *n*-dimensional discrete Cartesian space.

By CARPET, a variety of cellular algorithms can be designed in a simple but very expressive way. The language utilizes the control structures, the types, the operators and the expressions of the C language and it enhances the declaration part allowing specifying the features of a cellular automaton. These are the dimensions of the automaton (e.g., the declaration `"dimension 3;"` defines a three dimensional automaton), the radius (`radius`) of the neighborhood, the type of the neighborhood (`neighbor`). For example, a very simple neighborhood composed of only two cells can be defined as follows:

```
neighbor Couple[2] ([-1,0] Left, [1,0] Right);
```

As mentioned before, the state (`state`) of a cell is defined as a set of typed substates that can be *shorts*, *integers*, *floats* and *doubles* or *arrays* of these basic types. In the following example, the state consists of three substates.

```
state(float speedx, speedy, energy);
```

The `energy` substate of the current cell can be referenced by the predefined variable `cell_energy`. The `neighbor` declaration assigns a name to specified neighboring cells of the current cell and allows such to refer to the value of the substates of these identified cells by their name (e.g., `Left_energy`). Furthermore, the name of a vector that has as dimension the number of elements composing the logic neighborhood it must be associated to *neighbor* (e.g., `Couple`). The name of the vector can be used as an alias in referring to the neighbor cell. Through the vector, a substate can be referred as `Couple[i]_energy`. To guarantee the semantics of cell updating in cellular automata the value of one substate of a cell can be modified only by the `update` function (e.g., `update(cell_speedx, 13.3);`). After an `update` statement, the value of the substate, in the current iteration, is unchanged. The new value does take effect at the beginning of the next iteration. Furthermore, a set of global parameters (**`parameter`**) describes the global characteristics of the system (e.g., the permeability of a soil). CARPET allows to define cells with different transition functions (*inhomogeneous CA*) by means of the **`GetX`**, **`GetY`**, **`GetZ`** functions that return the value of the coordinate X, Y, and Z of the cell in the automaton. Varying only a coordinate it is possible to associate the same transition function to all cells belonging to a plane in a three dimensional automaton.

The language does not provide statements to configure the automata, to visualize the cell values or to define data channels that can connect the cells according to different topologies. The configuration of a cellular automaton is defined by the graphical user interface (UI) of the CAMEL environment. The UI allows, by menu pops, to define the size of the cellular automata, the number of the processors onto which the automata must be executed, and to choose the colors to be assigned to the cell substates to support the graphical visualization of their values. The exclusion from the language of constructs for configuration and visualization of the data allows executing the same CARPET program with different configurations. Further, it is possible to change from time to time the size of the automaton and/or the number of the nodes onto which the automaton must be executed. Finally, this approach allows selecting the more suitable range of the colors for the visualization of data.

In latest years, other cellular programming languages such as CELLANG [5], CDL [6], CARP [7], CEPROL [11] have been defined. These languages allow, as in CARPET, the definition of cellular algorithms. However, none of those contains all the features of CARPET, neither, at our best knowledge, a parallel run-time support has been implemented for them.

4. Designing a soil contamination model

In the design of a CA for soil contamination-decontamination, we refer to the contamination of a porous soil caused by a phenol rich water solution. This problem involves two phases only (water and air) while of course in other cases it might be necessary to consider additional phases (e.g. an oil phase in the case of hydrocarbon contamination).

Several attempts to use CA for modeling macroscopic physical phenomena have dealt with a microscopic approach, where the state variables can take a limited set of discrete values. For instance, the state of a cell represents a particle with spin 'up' or 'down'. In CABOTO, like in other simulations performed by CAMEL, a different approach was taken considering the cells as portions of space that include several pores. It is therefore meaningful to assign to each cell a more complex state, which is the Cartesian product of different substates, which may refer, for example, to water content, contaminant concentration, etc. Some of those variables are usually considered as continuous, and we admit their use. The formal compliance with the CA definition can be easily assured by resorting to a discretized set of values. Macroscopic CA models can be regarded as intermediate between classical PDE models and the so called compartment models, which provides a simpler process description by dividing the space into cells of finite size which are assumed to be in local equilibrium.

The cellular automaton used in our model is three dimensional and the cell's neighborhood is composed of the `up`, `north`, `east`, `west`, `south`, and `down` neighbor cells (see figure 1).

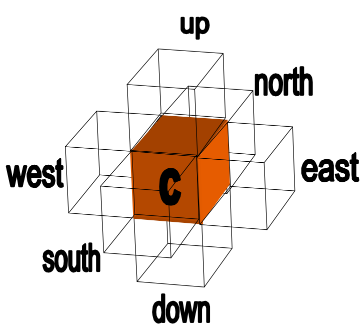

Fig. 1. The six cells that compose a cell (**C**) neighborhood.

We show how a designer can design the transition function that defines the specification of a soil contamination model by the CARPET language. The cell state and the neighborhood of the three dimensional automaton are defined in figure 2 that shows the declaration part of the model.

```
#define FALSE 0
#define TRUE 1
cadef
{
 dimension 3;
 radius 1;
 state (float Water, Porosity, OutFlow[6], KSat, CapThr,
              ConPhS, ConPhW);
 neighbor Neum3D[6](    [0,0,1]Up, [0,1,0]North, [1,0,0]East,
                        [-1,0,0]West, [0,-1,0]South, [0,0,-1]Down);
}
```

Fig. 2. CA declaration part.

The finite set of substates of the elementary automaton have the following meaning:

- *Water* is the water content in the cell.
- *Porosity* is the *effective porosity* parameter. It depends on physical-chemical characteristics of the soil in the cell and defines the quantity of water drainable from the cell in condition of full saturation, i.e. the porous volume that can be filled by water. For example, a cell with completely impermeable soil inside has the *effective porosity* parameter equal to zero, while the *effective porosity* of a cell is greater if the soil is more permeable.
- *OutFlow* represents the water flow toward the six neighborhood directions from the central cell. Note that the inflows are not explicitly considered, they are obtained trivially by the outflows.
- *KSat* is the hydraulic conductivity at the saturation, considering that inside the cell the soil is homogeneous.
- *CapThr* is the *capillary water threshold* substate, i.e. the maximum cell water, which is effected by the capillary forces.

- *ConPhW* accounts for the phenol concentration in the cell water.
- *ConPhS* accounts for the phenol mass adsorbed in the cell per unit mass of dry solid.

Here we describe the algorithms designed by CARPET of the deterministic transition function that implements the three-dimensional soil contamination model. The complete model of the system is composed of four layers as shown in figure 3.
Each layer is described by one of the following transition functions:

- ρ is the transition function of source cells that have a *Z* coordinate equal to 0, where water and solutes have *origin* at each time interval; the outflow to the inferior cell only is calculated according to the assigned feeding.
- σ is the deterministic state transition function for cells in a 3-dimensional automaton that describes the contamination phenomenon; a large outline of its specification will be given in the next subsections.

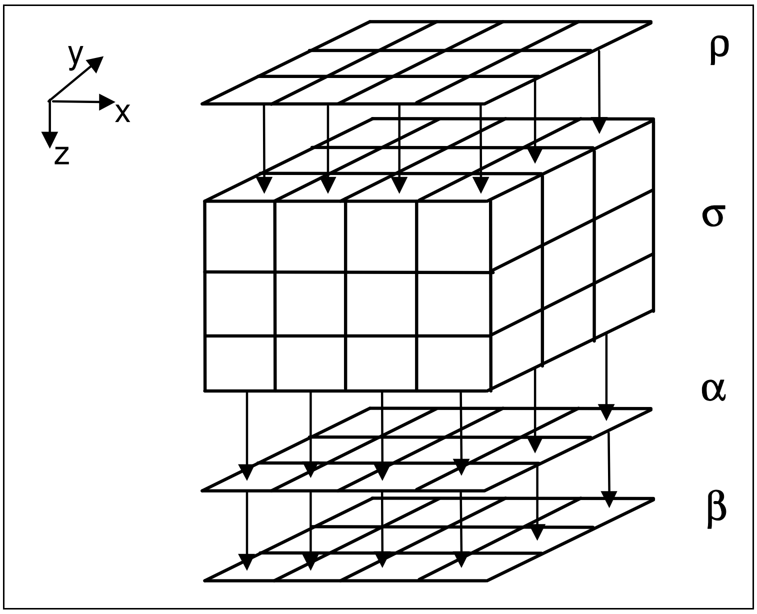

Fig. 3. Layers of the CA three-dimensional model of soil contamination.

- α is the deterministic state transition function for the cells in the first down layer, it specifies the behavior of ideal cells, which receive water only from the superior cell and transmit it to the inferior one without retention.
- β is the deterministic state transition function for the cells in the second down layer,

it specifies the behavior of ideal cells, which receive illimitable water without transmit it at all; it simulates water collection.

Such ideal cells with particular transition functions α and β are considered in order to describe forced conditions in laboratory experimental devices. At step 0 the initial configuration of the automaton is defined, specifying all the starting values of the cell substates. At each next step, the function σ is applied to all cells in the layer that describes the contamination phenomenon, while at the same time step the functions ρ, α, β are respectively applied to other layers, so the configuration is changed and the simulation of contamination is obtained.

4.1. The σ transition function

The σ transition function describes different phenomena, which interact in the system and can be grouped into two classes. The first class is the fluid dynamical one, which describes the flow of different fluid phases in a porous medium (the soil). In our case, only the flow of a gaseous phase (air) and a liquid phase (aqueous solution) have been considered. The second class describes the physical and chemical fate of the solutes. These phenomena are implemented using a transition function with two different sequential computational steps which act at each iteration.

The first step computes the new water substate values according to the following operations which must be executed sequentially in each cell:

1. first of all *cell_Water*, the new water content in the cell, is calculated considering the water balancing between current content *cell_Water*, inflows and outflows; inflows and outflows are determined by the contributions of capillary (*infc* and *outfc*) and gravitational (*infg* and *outfg*) inflows and outflows;
2. the new outflows determined by the capillary forces (*outfc*) are calculated;
3. the outflows determined by the gravitational forces (*outfg*) and the new global outflows (*outf*) are calculated.

The second step computes the solute flow through the soil.

4.1.1. The NewWaterContent procedure

The following CARPET procedure (figure 4) calculates the new water content according to the previous considerations and conventions; *SumW* is a variable used for the partial sums. The water outflows from a cell, or alternatively, part of the inflows into the adjacent cell it depends on the capillary and gravity forces. We will account on the main features of the transition function for the water flow.

```
NewWaterContent();
{   SumW = cell_Water;
    for (i = 0; i < 6; i++)
     SumW = SumW + infc[i] - outfc[i] + infg[i] - outfg[i];
    update(cell_Water, SumW);
}
```

Fig. 4. The NewWaterContent procedure.

4.1.2. The capillary water flow

Capillary forces involve a suction effect, which covers the porous surface with a water layer. The maximum possible thickness of the water layer depends on the attractive inter-molecular forces between water, air and soil.

We can characterize the equilibrium by a suction threshold (*cell_CapThr)*, which is the minimum water quantity inside the cell above, when the suction effects are practically negligible. Part of outflows (*outfc*) from the central cell to its neighbors is due to the balancing of the capillary forces, when the value of *cell_CapThr* is not reached in the neighbors. The capillary forces can be considered approximately proportional to the suction index *SuctInd,* here defined as (*1*-(*cell_CapThr* - *cell_Water*)/*cell_CapThr*), when *cell_Water* < *cell_CapThr* (0 otherwise). Thus, a balance can be given by outflows, which minimize the total difference of the suction index between a cell (the central cell) and its six neighbors after each time interval.

In the CARPET procedure, shown in figure 5, *Sum*W and *Sum*C*apThr* are variables for the partial sums of water content and suction threshold in the cells respectively.

```
NewCapillaryWaterFlow();
   {              /*  -------- action  1 ---------- */
     for ( i = 0; i < 6; i++)
      if (Neum3D[i]_Water < Neum3D[i]_CapThr)
       { eliminated[i] = FALSE;
         SuctInd[i] = 1 - Neum3D[i]_Water/Neum3D[i]_CapThr;
       }
      else
         eliminated[i] = TRUE;
      NewControl = TRUE;
      while (!NewControl)
      { NewControl = FALSE;
        SumW = cell_Water;
        SumCapThr = cell_CapThr;
        for ( i = 0; i < 6; i++)
         if (! eliminated[i])
          { SumW = SumW + Neum3D_Water[i];
            SumCapThr = SumCapThr + Neum3D[i]_CapThr;
          }
        AvSuctInd = 1 - SumW/SumCapThr;
        for ( i = 0; i<6; i++)
         { if ((SuctInd[i] > AvSuctInd) && (! eliminated[i]))
           { NewControl = TRUE;
             eliminated[i] = TRUE;
           }
         }
       }        /* ------- action  2 ----------*/
      for ( i = 0; i<6; i++)
       { if (!eliminated[i])
          outfc[i] = (1 - AvSuctInd) * Neum3D[i]_CapThr -
                        Neum3D[i]_Water;
         else
          outfc[i] = 0;
       }
   }
```

Fig. 5. The capillary water flow procedure.

Computation involves two sequential actions:

1. Selection of the neighboring cells (the so-called "eliminated cells"), toward which capillary water outflow is not possible, because their suction index is larger than the suction index of the central cell or because other neighboring cells have by comparison so low suction index that the capillary water flows necessarily only toward such cells.
2. Computation of the values of the outflows toward the not-eliminated cells in order to obtain the same value of the suction index in the central cell and in the not- eliminated cells.

4.1.3. The gravitational water flow

The gravitational water in a cell is the part of water *gw* which is practically not sensible to capillary forces. Its flow depends upon the hydraulic conductivity at the saturation, the porosity and the water content. In order to calculate the flow between the cell and its neighbor down, the conductivity value, which is considered, is the minimum of the conductivity of the two cells, *KMin*.

```
#define down 5
NewVerticalGravitationalFlow();
 {
  if (cell_KSat < Down_Ksat)
    KMin = cell_Ksat;
   else
    KMin = Down_KSat;
   MaxFlow[down] = gw * KMin;
   rec[down] = Down_Porosity - Down_Water-OutfSum;
   if (MaxFlow[down] > rec[down])
       outfg[down] = rec[down];
      else
      outfg[down] = MaxFlow[down];
 }
```

Fig. 6. The gravitational water flow procedure.

The CARPET code in figure 6 illustrates the algorithm for computation of the new vertical gravitational flow *outfg*. *MaxFlow* is the maximum possible gravitational outflow from the cell, *porosity* is the cell porosity, *rec* (receptivity) is the maximum water quantity which can penetrate a cell partially filled, and *OutfSum* is the sum of both capillary and gravitational outflows of a cell.

4.2. The solute flow through the soil

Cell total inflows (gravitational plus capillary) *inf* and outflows *outf* have been computed according to the previous algorithms.
Considering the phenol concentration *cell_ConPhW* in the cell water, the following algorithm (figure 7) calculates the new solute concentrations in the cell. *Phenol* is an auxiliary variable in order to find the phenol in the cell water, *res_w* is the residual water in the cell (considering only the outflows), and *NewW* represents the new water content in the cell. The water inflows in the cell can vary the solute concentration, altering the equilibrium conditions between the phenol concentration in solution *cell_ConPhW* and the concentration of the adsorbed phenol *cell_ConPhS*.

```
NewSoluteConcentrations ();
 ...........
  {
    res_w = cell_Water;
    for( i = 0; i < 6; i++)
      ResW = ResW - cell_OutFlow[i];
     NewW = cell_Water;
    for( i = 0; i < 6; i++)
      NewW = NewW + inf[i] - cell_OutFlow[i];
     phenol = cell_ConPhW * ResW;
    for( i = 0; i < 6; i++)
      phenol = phenol + inf[i] * Newm3D[i]_ConPhW;
    NewPh = phenol / NewW;
    update(cell_ConPhW, NewPh);
  }
```

Fig. 7. The solute flow through the soil procedure.

The value of the concentration of the adsorbed phenol at the equilibrium *EqCPhS* is computed, considering the Freundlich equation, with its two coefficients : the distribution coefficient *DistrCoeff* and the exponential coefficient *b. Solid* is the solid volume inside the cell. According to the clock of the CA, a rate, *SoluteRate*, is given in order to calculate how much is reduced the difference *DiffCPhS* between *EqCPhS* and *cell_ConPhS* after a step. The CARPET procedure in figure 8 illustrates the algorithm.

```
AdDeSorption();
 {
    EqCPhS  = Distr_Coeff * power(cell_ConPhW,b);
    DiffCPhS = (cell_ConPhS - EqCPhS) / SoluteRate;
    NewCPhS = cell_ConPhS - DiffCPhS;
    update(cell_ConPhS, NewCPhS);
    NewCPhW = ((cell_Water * cell_ConPhW) +
                     (DiffCPhS * solid)) /cell_Water ;
    update(cell_ConPhW, NewCPhW);
  }
```

Fig. 8. The AdDeSorption procedure.

5. Simulation and performance results

The model presented here has been simulated on a parallel computer. The simulation results of the first 8 hours of an experiment are reported in figure 9. The model captures a fluid-dynamical behavior, which is difficult to be simulated. The experimental results, reported here for comparison, have been performed in a pilot plant under carefully controlled conditions. The model involves a number of parameters, which can only partly be directly determined. The remaining parameter values have been found either by tuning them at hand, or by applying the novel optimization technique of genetic algorithms [10].

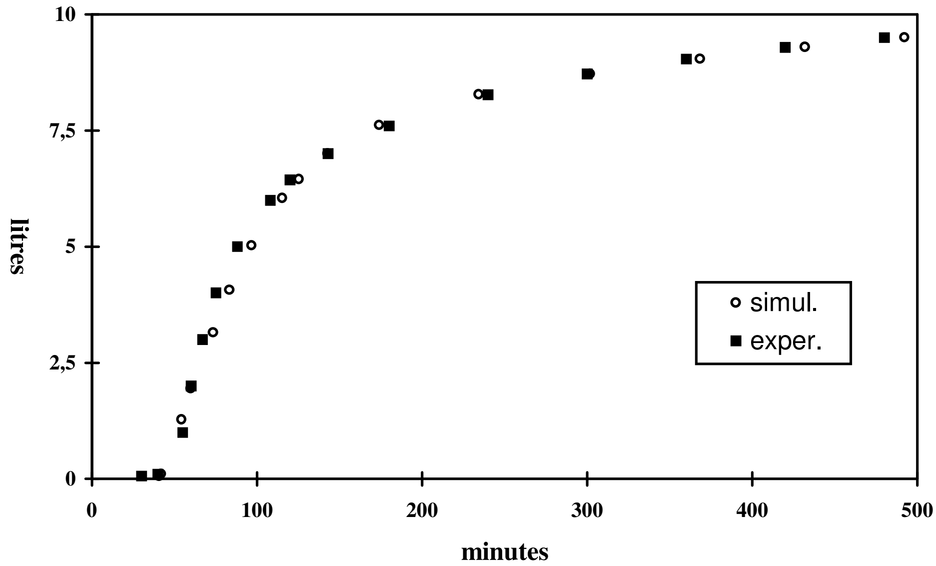

Fig. 9. The simulation data of water flow.

Figure 10 compares the experimental data of the phenol concentration in solution with the simulation results of about 50 days of an experiment.

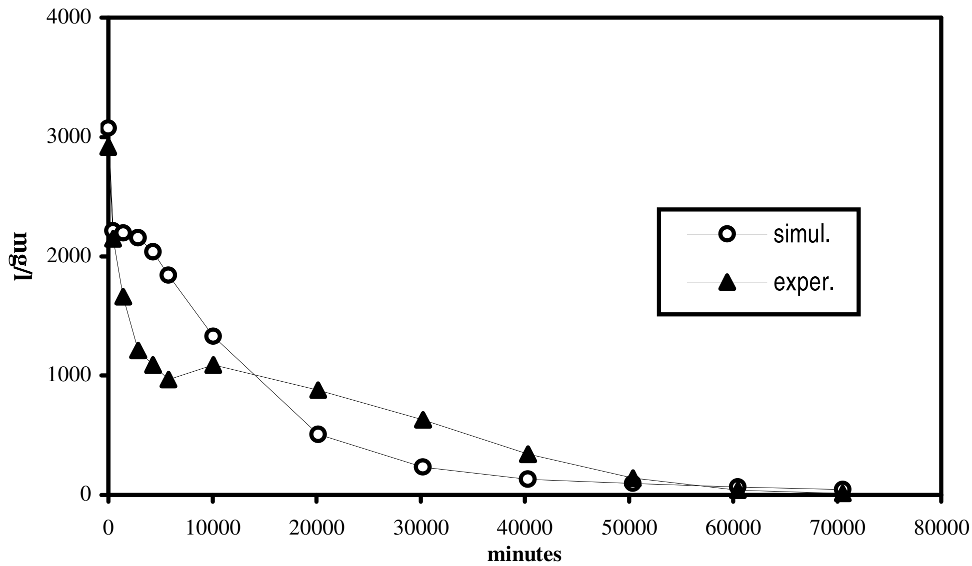

Fig. 10. The simulation data of drained water contamination.

The CAMEL system allows the user to see the animated visualization of a simulation by means of its graphical interface. Figure 11 shows a snapshot of the animated visualization of contamination simulation by the graphical interface of CAMEL.

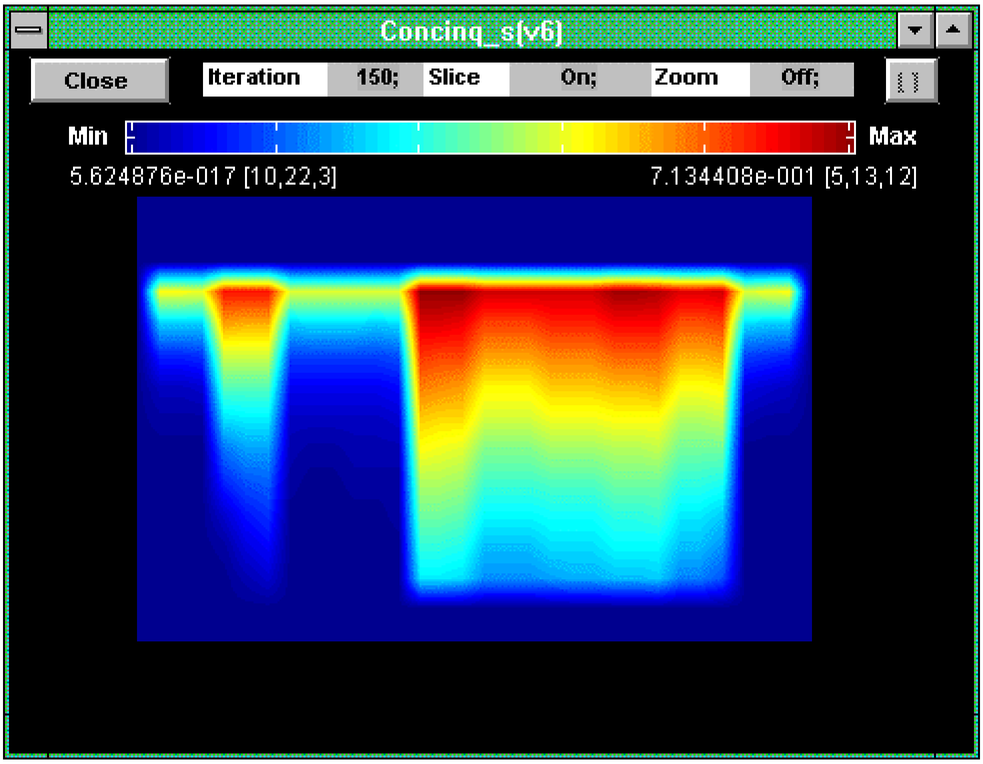

Fig. 11. A snapshot of the dynamical visualization of the contamination simulation.

We measured CAMEL execution time (in seconds) on our Transputer-based machine for different number of cells per processor, iterations and processor configurations. The models that have been simulated concern the water flow model only (*WF*) and the water flow plus contamination event model (*WF* + *C*). For comparison purposes, we show in figure 12 the speed-up measures for the two models. Figure 12 shows the speed-up results for a 32x17x13 (7072 cells) grid with 10 substates for the water flow model and 12 substates for the contamination event model. Comparing the results of the simulation with the experimental data, we deduce that 100 steps of the simulation correspond to about 12 minutes of the real phenomenon. Furthermore, we point out that the execution time of a single step is about 0,5 seconds. This result is very important because many iterations are necessary to complete our simulation on a long period basis (64 days). Consequently, it is worth noting that 24 hours of the real phenomenon can be simulated in about 1 hour and 40 minutes.

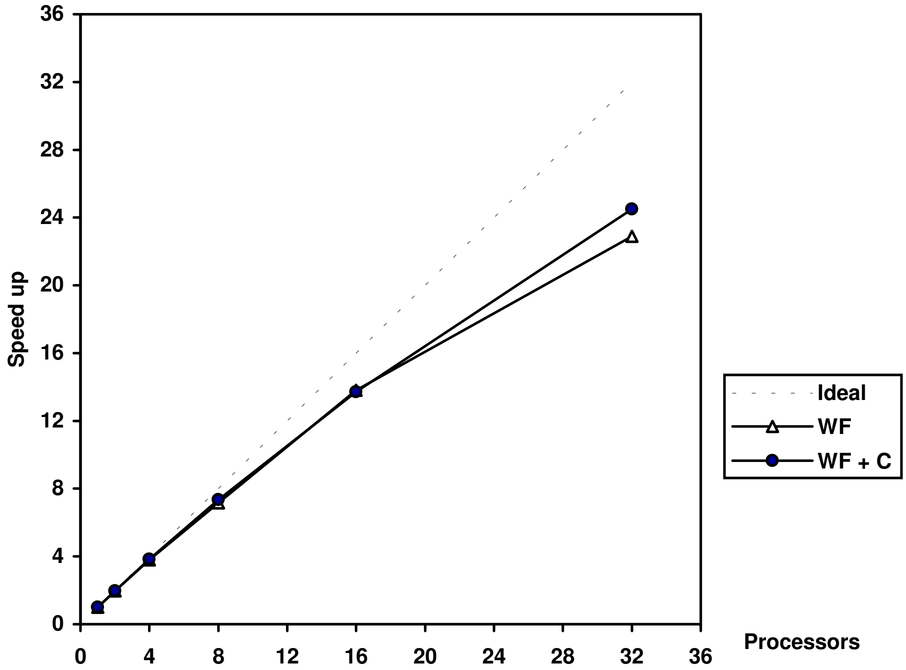

Fig. 12. Speed-up measures.

6. Conclusion and future work

In this paper a parallel design and implementation of a model of soil contamination using the CARPET language has been discussed in detail. The model is very useful in the study and evaluation of soil decontamination. The results of simulation fit quite well with experimental data and model performance is very interesting.

The major emphasis of the paper is about the use of a high-level language for programming cellular processing algorithms. The utilization of a language that explicitly defines data and constructs for programming cellular automata improved all the software development process. Through a real world application, we showed that

CARPET supports the development of complex algorithms based on the cellular automata paradigm.

After the successful experience of CABOTO, EC funded another ESPRIT project for the application of parallel computing to the simulation of the *in situ* bioremediation of contaminated soils. This new project is named COLOMBO (Parallel COmputers improve cLean up of sOils by Modelling BiOremediation). As an activity in this project, we are currently developing a portable MPI-based implementation of CAMEL and CARPET on MIMD parallel computers such as the QSW CS-2, the IBM SP2 and workstation clusters. Those powerful and up-to-date architectures will significantly increase the performance of our system.

The new system version is being developed using the MPI library because it is emerging as a de facto communication standard for MIMD parallel machines, in order to facilitate the portability to different hardware platforms (present and future). Portability is a crucial point related to the exploitation of parallel computing software. In particular, the new implementation of CAMEL and CARPET that will allow the end users to design and run simulations the on several parallel computers, besides the possibility to use the them on future parallel architectures.

Acknowledgements

This research has been partially funded by CEC ESPRIT P.C.I. contract n° 94529419370. We thank S. Di Gregorio, R. Serra, R. Rongo, W. Spataro and M. Villani for their work in the model definition.

References

[1] M. Cannataro, S. Di Gregorio, R. Rongo, W. Spataro, G. Spezzano, and D. Talia, A parallel cellular automata environment on multicomputers for computational science, *Parallel Computing*, **21** (1995) 803-824.

[2] B. Chopard and M. Droz, Cellular automaton model for the diffusion equation, *Journal of Statistical Physics*, **64** (1991) 859-892.

[3] S. Di Gregorio, R. Rongo, W. Spataro, G. Spezzano, and D. Talia., A parallel cellular tool for interactive modeling and simulation, *IEEE Computational Science & Engineering* **3** (1996) 33-43.

[4] S. Di Gregorio, R. Rongo, W. Spataro, G. Spezzano, and D. Talia, A parallel cellular environment for high performance scientific computing, in: H. Liddell et al., eds., *HPCN'96 High-Performance Computing and Networking*, LNCS 1067 (Springer-Verlag, Berlin, 1996) 514-521.

[5] J. D. Eckart, Cellang 2.0: reference manual", *ACM Sigplan Notices*, **27** (1992) 107-112.

[6] C. Hochberger and R. Hoffmann, "CDL - a language for cellular processing, in: *Proc. 2nd Intern. Conference on Massively Parallel Computing Systems* (IEEE Computer Society Press, 1996).

[7] G. Junger, "Cellular automaton tool user manual", GMD, Sankt Augustin, Germany, 1994.

[8] T. Karapiperis and B. Blankleider, Cellular automaton model of mass transport with chemical reactions, Bericht Nr.93-06 Paul Scherrer Institute, Villingen,1993.

[9] C.M. Pancake and D. Bergmark, Do parallel languages respond to the needs of scientific programmers?, *Computer,* **23** (1990)13-23.

[10] R. Serra and G. Zanarini, *Complex Systems and Cognitive Processes* (Springer, Heidelberg, 1990).

[11] F. Seutter, CEPROL: a cellular programming language, *Parallel Computing*, **2** (1985) 327-333.

[12] D.B. Skillicorn and D. Talia, Models and languages for parallel computation*, ACM Computing Surveys* (1998) (to appear).

[13] G. Spezzano and D. Talia, A high-level cellular programming model for massively parallel processing, in: *Proc. 2nd Int. Workshop on High-Level Programming Models and Supportive Environments (HIPS97)* (IEEE Computer Society, 1997) 55-63.

[14] T. Toffoli, Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics, *Physica 10D* (1984)117-127.

[15] R. van Liere, J.D. Mulder, and J.J. van Wijk, Computational Steering, *Future Generation Computer Systems*, **12** (1997) 441-450.

[16] J. von Neumann, *Theory of Self Reproducing Automata*, University of Illinois Press (1966).

[17] S. Wolfram, Computation theory of cellular automata, *Comm. Math. Phys.* **96** (1984).

[18] B.P. Zeigler, Y. Moon, D. Kim, and G. Ball, The DEVS environment for high-performance modeling and simulation, *IEEE Computational Science & Engineering*, **4** (1997) 61-71.