To appear in IEEE Computational Science and Engineering

A Parallel Cellular Tool for Interactive Modeling and Simulation

S. Di Gregorio*, R. Rongo*, W. Spataro*, G. Spezzano§ and D. Talia§

§*CRAI, Località S. Stefano, Rende (CS), Italy,*
Phone: +39 984 8331, Fax: +39 984 446044, Email:{gds,dot}@crai.it

**Dept. of Mathematics, Univ. of Calabria, 87036 Arcavacata (CS), Italy*
Phone: +39 984 493001, Fax: +39 984 401186

Version #2 September 1995

Abstract

In this paper we describe CAMEL, an interactive parallel environment based on the cellular automata theory. CAMEL is a tool designed to support the development of high performance applications in science and engineering. It offers the computing power of a highly parallel computer, hiding the architecture issues from a user. The system can be used both as a tool to model dynamic complex phenomena and as a computational model for parallel processing. CAMEL implements a cellular automaton as a SPMD (*Single Program Multiple Data*) program and includes a load balancing strategy to equally distribute the computation load over the processing elements. By CAMEL a user might write programs to describe the actions of thousands of simple active objects, then observe the global complex evolution that arise from all the interactions. The paper presents a set of applications in significant areas such as lava flow, landslides, freeway traffic, image processing, and genetic algorithms where CAMEL has been successfully utilized to solve complex problems.

Keywords: Cellular automata, complex systems, computational science, parallel computing, simulation.

1 Introduction

Computer simulation and modeling in the study of real physical phenomena provides a *virtual laboratory* where to investigate very complex problems in many areas such as biology, physics, chemistry, economics, and engineering.

For many years it has been very difficult to study the behaviour of complex phenomena because the formal models describing them were so hard that the main computational modality, represented by the integration of differential equations, was intractable even using powerful parallel computers.

The development of computer science in the latest years considerably enlarged its application boundaries because of the continuous rise of computing power. At the same time research in parallel computing showed evidence of the significant potential of parallel computing models, such as cellular automata and neural networks, in representing a valid alternative to differential calculus in the description of complex phenomena [1]. This occurs especially when differential equations cannot efficiently be solved because of their complexity or when is very difficult to model the problem being solved in terms of differential equations.

Cellular automata (CA) are very effective in solving scientific problems because they can capture the essential features of systems in which the global behaviour arises from the collective effect of large numbers of locally interacting simple components [2]. A cellular automaton is a discrete dynamic system composed of a set of cells in a one-dimensional or multi-dimensional lattice. Each cell in the regular spatial lattice can have any one of a finite number of states. The states of the cells in the lattice are updated according to a local rule called *transition function*. That is, the state of a cell at a given time depends only on its own state in the previous time step and the states of its nearby neighbors at the previous time step. Different neighborhoods can be defined for the cells. All cells of the automaton are updated synchronously. Thus the state of the entire automaton advances in discrete time steps. The global behavior of the system is determined by the evolution of the states of all cells as a result of multiple interactions.

Introduced by von Neumann the 1950's to investigate self-reproduction [3], CA have been used mainly for studying parallel computing methods and the formal properties of system modelling. In particular in the latest years, scientists like Brinch Hansen, Shapiro and others have described how the CA model can be effectively used as a parallel architecture model and a programming methodology for computational science and engineering applications on parallel computers [4, 5].

Cellular automata have been used to solve a wide range of problems in many areas such as physics and chemistry (e.g., fluid turbulence [6]), geology (e.g., rock fracturing [7], landslides, and lava flows), biology, computer science, traffic flow, science of materials, economics, and social sciences.

After a study of the interesting features of the CA model and considering that highly parallel architectures represent the natural background where a CA-based environment could be implemented, we decided to implement a parallel CA programming environment called CAMEL to be used as a *virtual laboratory* for the simulation of real complex phenomena.

CAMEL (*Cellular Automata environMent for systEms modeLing*) is an interactive tool designed to support the development of high performance applications in science and engineering. It offers the computing power of a parallel computer hiding the architecture issues from a user. CAMEL can be used both as a tool to model dynamic complex phenomena and as a computational model for parallel processing. Other CA-based parallel environments that have been implemented in the latest years are CAM[6], CAPE [7], and StarLogo[8].

Here we describe CAMEL and its usage for programming models and simulations of complex real phenomena. The main focus of the paper is on the development of applications by CAMEL. The main part of the paper discusses the CAMEL programming approach and a set of scientific and engineering application areas such as geology, fluid dynamics, traffic flow, image processing, and genetic algorithms where CAMEL has been successfully utilized to solve significant problems. In fact, our aim is to describe how CAMEL offers a novel and very interesting approach for developing complex applications in several fields of science and engineering allowing the interdisciplinary cooperation of people working in different research areas. Before to discuss the CAMEL applications we give an essential description of the design of the architecture of CAMEL. This allows the reader to know how the CAMEL components have been organized to reach the system goals.

2 CAMEL architecture

CAMEL [9] exploits the computing power of a highly parallel computer hiding the architecture issues from a user in programming computational science applications. It provides a user interface that allows a user to interactively change the parameters of a simulation at run time, and a graphical interface that provides an animated visualization of the output of the application.

We have chosen to implement CAMEL on an affordable message-passing MIMD parallel computer. It is an INMOS ITEM-4000 parallel computer composed of a thoroidal mesh of 32 Transputers connected to a host node composed of a PC with a Transputer board. This parallel computer makes the CA environment a scalable system where no bottleneck will emerge when the number of processors is increased or when the size of an automaton is scaled up. Further, MIMD computers are more flexible to implement CA in comparison with SIMD machines. Finally, the Transputer-based machine is cheaper in comparison with other very expensive parallel computer such as the CM-5, the CRAY T3D or the Intel Paragon and it has a good performance/cost ratio.

The current implementation of CAMEL does not limit the number of processing elements (PEs) that can compose the parallel computer, so no relevant changes are necessary in the software of CAMEL whether a very large number of PEs should be used. Furthermore, the CAMEL architecture is suitable to be implemented on others distributed memory parallel computers such as the Intel iPSC, the CM-5, and the CRAY T3D.

2.1 Parallel Architecture

The parallel architecture of CAMEL comprises three main components, the *Master Node*, the *Graphic Node*, and the *Parallel Engine*. In particular, the *Parallel Engine* consists of a number of identical nodes on which the cellular automaton is executed. Figure 1 shows a scheme of the architecture of CAMEL.

The *Master Node* is implemented on a single Transputer. The *Master Node* is composed on the Controller (CNT), the User Interface (UI), and a Multiple Ring Router (MRR) processes.

On each Transputer that composes a node of the *Parallel Engine* runs a Macrocell (MC) process and a MRR. The MC is the heart of CAMEL. An MC implements a cluster of elementary cells that constitute one partition of a cellular automaton. Each MC executes the state transition function defined by a user for all the cells that compose the partition. The whole set of the MCs implements a cellular automaton.

On the *Master Node*, the CNT provides a coordination among the Macrocell processes, the file system, and the UI. The main task of the CNT process is the synchronization of the Macrocell processes that are running on the nodes of the *Parallel Engine*. The UI allows a user to monitor the parameters of a system simulation and to dynamically change them at run time. A detailed description of the UI is given below.

The collection of the MRRs implement a message routing strategy that allows message communication between two Macrocell processes regardless of where they are physically located on the *Parallel Engine* and between a Macrocell process and the *Master Node*. The routing algorithm has many positive characteristics including provable deadlock-freedom, guaranteed message arrival, and automatic local congestion reduction.

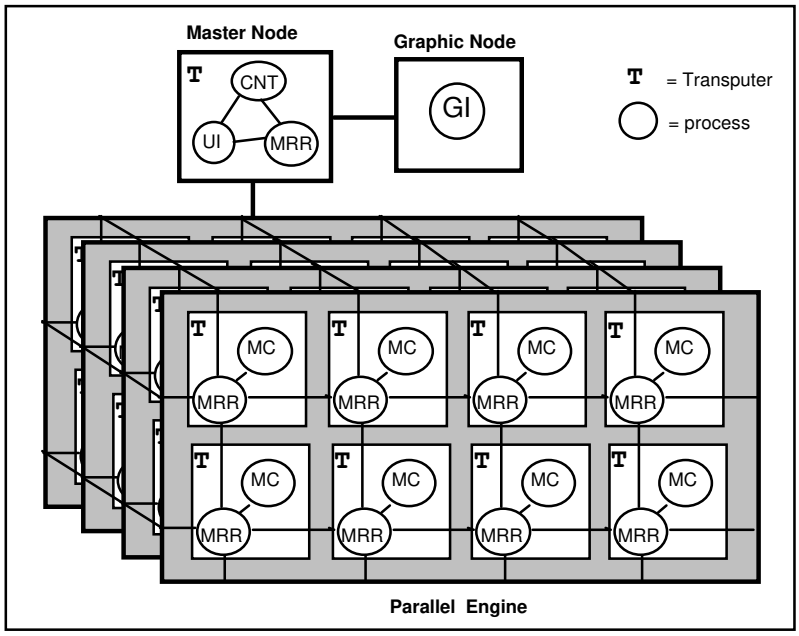

Figure 1. The CAMEL architecture.

The *Graphic Node* is implemented on a Transputer graphical board. On it is mapped the Graphical Interface process (GI). The GI displays on a screen the state of a system (the set of the states of cells) which is simulated by CAMEL.

In CA applications, cells with passive states don't change state until the state of neighbor cells does not change too, while cells with inert states never change state. For a large class of problems, as in flow diffusion, the areas of active cells are restricted to one, or few domains. From an computational point of view is useless to compute the next state of the passive cells until they become active. Thus, it is important to develop a load balancing strategy to manage the stationary regions in order to save precious computational power.

We implemented in CAMEL a load balancing strategy [9] that avoids to compute the next state of cells that belongs to a stationary region. Further, the load balancing strategy divide the computation of the next state of the active cells among all the processors of the parallel machine. The load balancing strategy used in CAMEL is a domain decomposition strategy similar to the scattered decomposition technique. Our strategy is applicable to MIMD parallel

computers and it is suitable for problems where the load may vary greatly between processors (e.g., in a dynamical simulation or where an irregular region is being modelled).

2.2 User Interface and Visualization

In complex problem programming, graphic output and animation represent a necessary tool for a rapid and clear reading of the large set of numeric values that result from the computation. Thus high level and user friendly tools allowing a user to interact with the system are a basic part of the environment.

The User Interface of CAMEL has been implemented in the MS-Windows environment and provides a window-oriented graphical interface that allows to perform an interactive simulation (figure 2). In such a way the UI offers to a user an integrating environment that allows to perform all the steps needed to develop an application in CAMEL, from the program editing to the configuration, running and monitoring of a simulation. In particular, for the configuration 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 colours to be assigned to the cell substates for the graphical visualization of their values. The UI provides useful information such as the displayed substates, the current iteration step, the visualization step, and the step of saving. Further, the UI allows a user to change the system parameters during the application running. This feature is used for monitoring and tuning the application. It also allows to adapt the visualization of the simulation output in order to simplify the user interpretation of the application results.

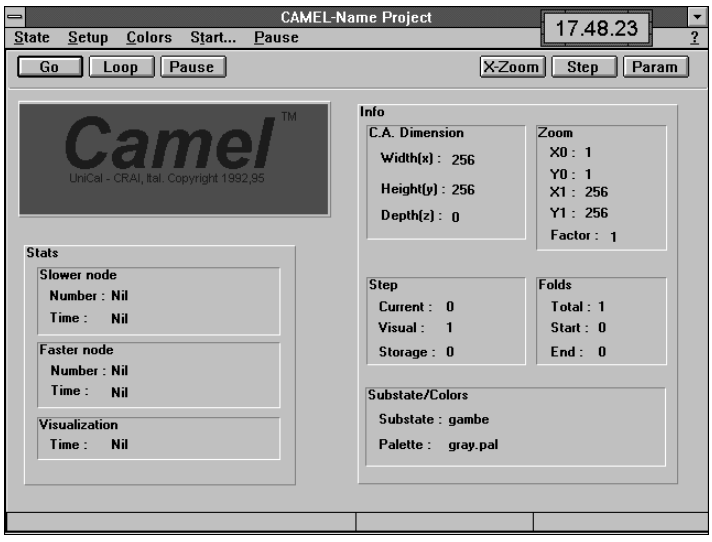

Figure 2. The main window of the CAMEL User Interface.

The GI visualizes the output of a simulation in a graphic format providing an animated visualization of the evolution of the system being simulated (see for example figures 7 and 8). This component receives raw data from the Macrocells on the network, commands from the UI and then performs the action needed to the on-line visualization. A range of values to be displayed and a corresponding range of colours to be used is associated to each substate.

A user can zoom a displayed image or a portion of it to obtain a better definition level. Besides, it is also possible to visualize more substates (e.g., temperature and fluid density), overlapped on the same window, or in different windows. This provides a more complete system evolution representation, even if it requires a higher visualization time.

3 CAMEL programming

Our main goal in the development of CAMEL was to implement a high-performance system to be used in a simple way by people coming from different scientific areas without a specific background in parallel computing. Thus, the rational of CAMEL is to make parallel computers available to application-oriented users hiding the implementation issues rising from their architectural complexity.

CAMEL allows the simulation of complex systems that may be represented as discrete across different lattices. In such systems, the values of a specific point at time $t+1$ are dependent upon the values at time t of its neighbor points (fig. 3) on the basis of a state transition function. Each system that can be specified using this approach can be modelled or simulated using CAMEL. CAMEL implements a cellular automaton as a SPMD (Single Program Multiple Data) program. In the SPMD model the same program is run on all the processors of an MIMD computer. Because an entire program is executed on separate data, the processors no longer perform the same operation in lockstep, as occurs in the SIMD model, but at a given time they might execute different instructions within the same program.

According to the SPMD approach, CA are implemented as a number of medium-grain processes each one mapped on a distinct processing element that executes the same program on different data (a partition of the cell lattice).

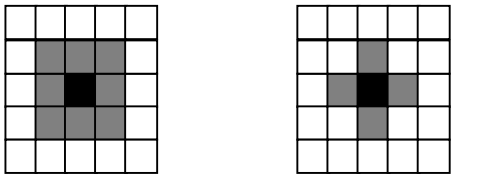

Figure 3. Examples of two different neighbourhoods in a square lattice.

The main components of an application definition are the lattice size, the state of cells, the local rule (the state transition) that specifies the behavior of the system, and the type of neighbourhood. According to this approach, users, can specify the transition function that governs the system they wants to model by a program written using the CARPET (CellulAR Programming EnvironmenT) language. CARPET is the programming tool by which it is possible define cellular programs within the CAMEL environment. It is a high-level language based on C language with additional constructs to describe the rules of the transition function of a single cell in the lattice. The main features of CARPET are the possibility to describe the cell state as a structured variable composed of typed fields (substates), and the simple definition of complex neighbourhoods (e.g., hexagonal, Margolus, etc.), that can be also time dependent. In the CARPET program are defined the basic rules of the system to be simulated and the macroscopic quantities that constitute the global parameters of the model, but nothing needs to be specified about the parallel execution.

The main features of CARPET are briefly described here, by a simple program that illustrates the HPP lattice gas proposed by Hardy, de Passis, and Pomeau to model a fluid of particles (fig. 4). This two-dimensional lattice gas uses a von Neumann neighborhood with only a single speed of particle. Each cell stores information on up to four particles. Each particle must be traveling in one of the four directions. Thus, no more than one particle can be incident to a cell from each of its four neighbors. Normally, particles pass through a cell unaffected. However, when exactly two particles collide head-on within a cell, deflect one another by right angles.

In the example showed in figure 4, by the `cadef` section we defined a two-dimensional automaton where the state of cells is composed of a single substate (`particle`) and the neighborhood is composed of 4 cells. The substate `particle` of the current cell or of a neighbor cell can be accessed appending it to the predefined variable `cell` or the identifiers of the neighbor cells (e.g., `cell_particle, north_particle`). The `update` function updates the value of the state of the current cell to be used in the next iteration step.

```
/* The value 0 indicates no particles and state values 1..15 indicate */
/* the presence of particles traveling in the north (=1), east (=2),  */
/* south (=4), and west (=8). These values are added to get all the   */
/* particle directions. For example if a cell has both a east and a   */
/* south bound particle its state value would be 6.                   */

cadef
{
 dimension 2;
 radius 1;
 state (short particle);
 neighbor Neumann[4] ([0,-1]North, [-1,0]West, [0,1]South, [1,0]East);
}

short FromNorth, FromSouth, FromWest, FromEast;
short sum;

FromNorth = FromSouth = FromWest = FromEast = 0;
if (North_particle>11) || (North_particle>3 && North_particle<8)
  FromNorth = 4;

if (South_particle % 2 == 1)
  FromSouth = 1;

if (East_particle>=8)
  FromEast = 8;

if (West_particle==2  || West_particle==3  || West_particle==6  ||
    West_particle==7  || West_particle==10 || West_particle==11 ||
    West_particle==14 || West_particle==15 )
  FromWest = 2;

if (FromNorth==4 && FromSouth==1 && FromEast==0 && FromWest==0)
   update (cell_particle,10);
else
 {
  if (FromEast==8 && FromWest==2 && FromNorth==0 && FromSouth==0)
   update(cell_particle,5);
  else
    {
     sum = FromNorth + FromSouth + FromEast + FromWest;
     update (cell_particle,sum);
    }
 }
```

Figure 4. The HPP lattice gas automata implemented with CARPET.

Generally, a program that implements a transition function is composed of few lines for simple applications or a few hundred lines for complex applications. When the program has been compiled and linked to the CAMEL support, its code is automatically mapped on each

computing node and it is executed in parallel to update the state of each cell. For each cell the transition function is executed repeatedly in a loop. A user may then observe on the graphical screen the system evolution and eventually change the parameters or stop the simulation by means of the User Interface.

4 Applications

This section presents five real world applications that have been developed utilizing CAMEL. The heterogeneity of the applications show the versatility of CAMEL and the expressiveness of the CA paradigm both as a tool to model dynamic complex phenomena and as a computational model for parallel processing. The applications presented here have been developed by an interdisciplinary effort where computer scientists cooperated with geologists, engineers, mathematicians and physicists. Each member gave a personal contribution based on its particular background to solve the complex problems discussed here.

The first two applications model geophysical phenomena. They regard the simulation of lava flows and landslides. These phenomena are very difficult to model by differential calculus because it is very complex to solve the fluid-dynamic equations with a changing viscosity and a real morphology.

The approach adopts methods of *qualitative* physics and doesn't consist in the resolution of a model described by Navier-Stokes equations, though it is possible by CA to perform a discrete simulation of a system of partial differential equations. The approach is based on modelling the most important aspects of the phenomenon, introducing also, whether useful, parameters that might not have a direct correspondence in physics. For example, in the case of the lava flow simulation the transition function that describes the evolution of the phenomenon utilizes as a parameter the *adherence* (that rules the viscosity), which has no correspondence in physics.

The possibility to implement a tool that performs careful simulation of those phenomena is of considerable interest because it can be used as a decision support system in territory control and for planning interventions to minimize the risks.

The third application shows the use of CAMEL for microscopic freeway traffic modeling. This approach is based on the simulation of the behavior of individual vehicles. Having a large set of vehicles and a real freeway, this problem can be solved only using a high

performance parallel computer. We show how CAMEL has been useful to accomplish this task.

The last two applications utilize CA as a parallel computation model. In the first case the CAMEL environment is utilized to develop parallel image processing software. The parallel implementation of the algorithms regards low level image processing (edge detection, filtering, smoothing, etc.) where it is necessary to execute simultaneously simple operations on all the elements (pixel) that form the image and the size of the final image is the same as that of the initial image.

The last application is the implementation of parallel genetic algorithms (GA) that are a stochastic general-purpose technique of *search* inspired by natural evolution. The model utilized for parallel implementation of GA is fine-grained and is known as *diffusion* model. The main advantages of this model are the possibility to use a large population to improve the quality of the search and the parallel evaluation of the *fitness* associated with each element of the population.

To program these applications by CAMEL we only needed to design for each different application a CARPET program that implements the operations of the state transition function of the cells that compose the lattice. In all the applications discussed here the system demonstrates to be scalable. For example, figure 5 shows the speedup results obtained for the lava flow and landslide simulations. In particular, the speedup on 32 PEs is respectively 28.1 in the lava flow case and 25.0 in the landslide case.

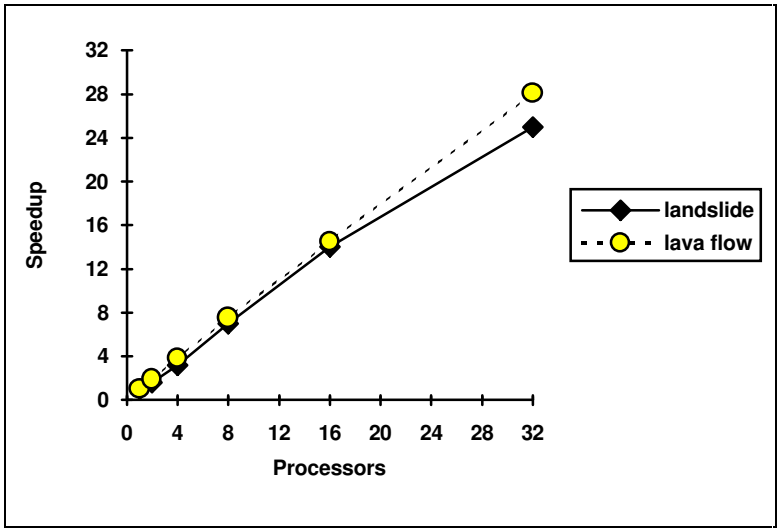

Figure 5. CAMEL speedup results.

The behaviour of the current CAMEL prototype has been very satisfying for the various applications implemented on it both in terms of speedup and modeling thoroughness. Currently, CAMEL is used as development tool in CABOTO (Cellular Automata for the Bioremoval of Toxic Contaminants), a research project supported by the European ESPRIT programme. The main goal of CABOTO is the application of cellular automata models to the bioremediation of contaminated soils and the development of a software environment for their simulation on high-performance parallel computers. In the next future other application areas in science and engineering will be explored and CAMEL will be used as virtual laboratory for solving complex problems in these fields.

4.1 Lava flow simulation

An important goal in volcanology is the ability to simulate eruptive phenomena for reliably forecasting their potential hazard. In this area the forecast of lava flows due to a volcanic eruption are one of the phenomena more studied. In this section we briefly present a CA model for simulating lava flows [9] and its application for the simulation of the 1986-87 and 1991-92 Etnean eruptions.

The CA model for simulating lava flows can be viewed as a dynamic system based on local interactions with discrete time and space, where the space is a two-dimensional lattice partitioned into square cells of uniform size. Each cell represents a portion of land surface and it is characterized by specific values (the state) of the following selected physical parameters: altitude, lava thickness, lava temperature and lava outflows toward the neighbouring cells.

The initial configuration of the automaton can be obtained from both pixel-based satellite images of the landscape and with quadrat-based field observations. The transition function of each cell evolves by computing the lava outflows toward the neighbouring cells and updating the temperature value. The temperature is modelled by a two-step process which describes changes due to lava motion through a cell and to thermal energy loss from the cell surface. The new cell thickness at time $t+1$, defined by the substate *thick*, is given by the thickness at time t plus the lava inflows (`Neumann_inflow[i]`) from the neighbour cells minus the lava outflows (`Neumann_outflow[i]`) from the cell toward its neighbour cells. The cell altitude may change when lava begins to solidify. When the cell temperature becomes equal or less than the solidification temperature of lava, the altitude is increased by the lava thickness of the cell. At the same time the lava thickness of the cell is reset to zero.

The section of the transition function that concerns computing of the new value of the tickness and the solidification effect are shown in figure 6.

```
                                    ....
/* computing new value of tickness */

newtick = cell_tick;
for (i=0; i <4; i++)
newtick = newtick + Neumann_inflow[i] - Neumann_outflow[i];
update(cell_tick, newtick);

/* effect of solidification */

if (cell_temperature <= Tsol)
   {
     newalt = cell_altitude + cell_tick;
     update(cell_altitude,newalt);
     update(cell_tick,0);
   }
                                    ....
```

Figure 6. A fragment of the transition function computing the new value of tickness and the solidification effect.

To take into consideration the rheological resistance, a parameter called *adherence* has been introduced to represent the amount of lava (expressed as a thickness) that does not flow outside the cell. Outflows are computed according the principle of communicating vessels.

The model was validated by the simulation of 1986-1987 Etnean lava flows. Then, the parallel simulation by CAMEL [9] has been used on a real emergency in Sicily when a lava flow threatened the town of Zafferana Etnea in April and May of 1992 (fig. 7).

Figure 7. A snapshot of the Etna's lava flow simulation.

The automaton is composed of 130560 cells. To obtain the final shape of the lava flow where needed 6000 steps. The time necessary to complete the simulation was about 1 hour and 45 minutes without the use of the load balancing strategy and 1 hour and 10 minutes using load balancing.

4.2 Landslide modeling

Our main goal in designing a landslide model was to develop a tool allowing the running of simulations of landslides with different physical characteristics. This tool should be used as a 'soft' environment to support the prediction of landslide behaviour and for planning in situ interventions when a landslide occurs.

The CA model for debris flow is similar to that utilized to simulate lava flow. Each cell of the two-dimensional plane represents a portion of land surface, in which the altitude, the thickness of the debris, the water content in the debris, and the debris flow toward the neighbourhood cells compose the state of the cell. The state evolution depends on a transition function, which simulates the physical processes of the present debris flow. At the beginning, the morphology of the landscape is introduced assigning to each cell a value to the substate altitude. The substates that represent the thickness of mobile land and the water content in the debris are initialized only for the cells that represent the portions of the surface from which the landslide comes away. The substates for the other cells are initialized to zero.

The transition function is deterministic and its behaviour can be briefly described as follows. The cell altitude is increased by the thickness of debris once the debris water content drops to a minimal value so that the motion is blocked (solidification). At the same time the debris thickness of the cell is reset to zero. When the debris thickness is larger than a critical value and there is no solidification, the cell altitude is decreased by a spooning factor proportional to the difference between the debris thickness in the cell and a critical value. At the same time the thickness is increased by the spooning factor.

The debris thickness in a cell at time $t+1$ is given by the thickness at time t adding debris inflows from the neighbor cells and subtracting debris outflows to the neighbor cells. The *adherence* parameter describes the quantity of debris that cannot flow to the neighbouring cells because of the low water content. Changes in cell water content are modelled as a two-step process to describe effects due to mixing into the cell of debris with different water content and to water loss at the ground-debris surface.

The validation of this model has been carried out on a real case simulating the Ontake volcano landslide that occurred in Japan in 1984. The results obtained (fig. 8) have showed the soundness of this approach as to other described in literature. To obtain the final shape of the landslide where needed 6000 steps (fig. 8.d). The simulated landslide is very similar to the original landslide (fig 8.e). The debris path is well identified, but the simulated flow widths are, at the beginning, larger (the cohesion effect is not considered) and, on following, narrower as an effect of the initial situation because the flow is smaller even if longer in the simulation.

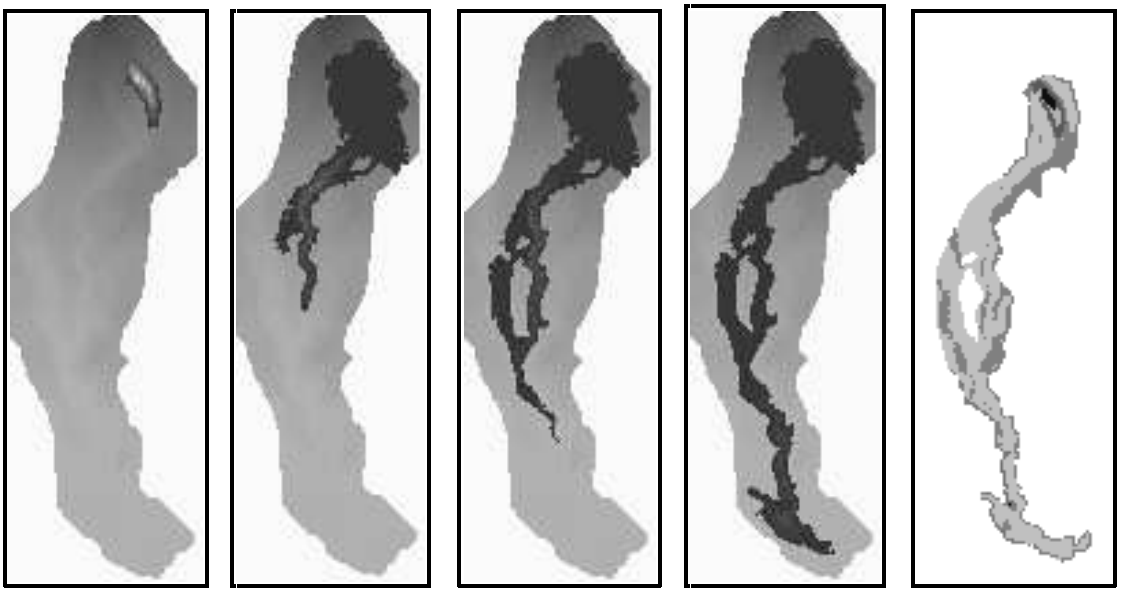

Figure 8. The Ontake mountain landslide simulation.

The automaton is composed of 15680 cells. In this case the time necessary to complete the simulation on 32 PEs has been about 35 minutes without the use of the load balancing strategy and about 24 minutes using load balancing.

4.4 Freeway traffic simulation

Traffic flow modeling represents a high complex problem. Non-linear dynamics methods based on classic differential equation methods were applied to model traffic flow in order to manage only partial aspects of the phenomenon.

The high computing power of parallel systems might allow the implementation of *microscopic* road traffic models where each vehicle is a single entity able to adapt its own behaviour to the structural road characteristics and the mutual interactions among vehicles. We used the cellular

automata theory as formal support to develop a microscopic model for freeway traffic then we implemented the model using CAMEL.

Our freeway traffic model $A_{freeway}$ is microscopic and it is based on the following elements: structural freeway characteristics, vehicle features, traffic condition and mutual interaction among vehicles [10]. A single cell of the automaton represents a segment of freeway (one way only) five metres in length equivalent to the average vehicle length and its state is composed of the elements mentioned before.

The freeway is composed of three lanes l_1, l_2 and l_3, respectively the right, central and left one. Furthermore, another lane l_0 is added for entrance, exit and emergence.

The transition function is composed of two parts:

- The first one calculates for the vehicle in the cell the new values of actual speed and lane together with the shift (number of cells forward where the vehicle arrive in the next step), in order to reach the desired speed in safety conditions;
- The second explores back cells in order to find vehicles whose shift brings exactly in the cell, then the values of substates that describe the features of such a vehicle in the back cell will be copied in the current cell.

The following functions are components the transition function of each cell:

a) *Check_speed*, having the lane, the speed limit, next and selected exit as arguments, it returns the maximum speed (in safety conditions) which the vehicle can reach in the cell at that lane; it accounts the eventual vehicles immediately forward and backward in all the lanes in order to calculate the next position of the vehicles, considering lane changes too; the speed value is obtained comparing the next possible position of the vehicle at the same speed with speeds and all the next possible positions of the neighbour vehicles;

b) *Distance*, with the of lane as argument, it returns the distance from the vehicle immediately forward in the lane;

c) *Decelerate*, having the distance from the vehicle immediately forward, speed, and deceleration capacity of the vehicle, and the freeway gradient as arguments, it returns the new speed obtained by the least possible deceleration in optimal safety conditions;

d) *Accelerate*, using as arguments the distance from the vehicle immediately forward, the speed, the maximal possible speed and acceleration capacity of the vehicle, and the freeway gradient, it returns the new speed obtained by the largest possible acceleration in optimal safety conditions;

e) *Change,* calculates the vehicle shift having as arguments the old and new speed, the values of acceleration-deceleration capacity of the vehicle and the gradient.

The properties of the CAMEL environment allowed an efficient implementation of the $A_{freeway}$ model. The simulator implemented on CAMEL showed to be scalable, obtaining a speed-up, in comparison with a single processor, equal to 28 on 32 processors. It simulates up to 1200 vehicles in 0.25 times the real time (it takes 1 processing second to simulate 4 real seconds).

In the current implementation of the simulator the freeway length can reach 1000 kilometres. 25 kilometres appear on the screen, divided in 25 segments, 1 kilometre long. Three lanes and the access/exit/emergence lane are visualised. Three types of cars, occupying a cell, and two types of commercial vehicles, occupying two and four cells, are represented by different colours.

Figure 9 shows a snapshot of the simulation output visualized on the graphical screen. The snapshot shows three consecutive segments of the freeway where different vehicle types are displayed by different colours.

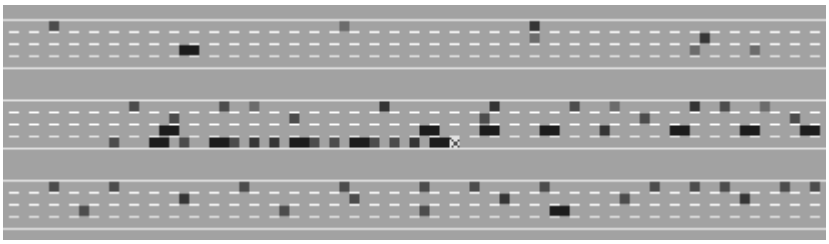

Figure 9. A snapshot of the simulated freeway visualization.

The model simulates correctly the global vehicle behaviour in the traffic different conditions also in merging and exit points. The perturbing agents such as obstacles correctly produce a vehicle queue. For example, the simulated drivers reduce their speed as the vehicular concentration increases in order to keep the safety conditions, obtaining correct relations between average speed traffic and traffic volume. Further, a first complete test on the model validity in conditions of free vehicular flow has been performed satisfactory. Simulation data fit well the standard field data found in the literature [10].

4.5 Image processing

Image processing is an application area where high performance computing is particularly effective. Problems in image processing require to process a large amount of data in very short time and need high speed computing that can be offered only by massively parallel computers.

In image processing there can be identified three different computational phases known as: low, intermediate and high level. The CA paradigm can be used as a computational model for parallel image processing in the low level phase. The main advantage, shared with other data-parallel models, in using CA for parallel image processing is that the processors simultaneously work on all pixels of the image executing only local operations in synchronous way. Image processing can be easily modelled by CA associating a cell to each pixel of the image. The array of the cells composing the automaton has the same size of the image that must be processed. The state values of a cell correspond to the values of luminance assigned to a pixel. In images with grey levels a pixel can take values from 0 to 255.

Each cell has a neighbourhood that is composed by an $n \times n$ window centred on it, typical values of n are 3, 5 and 7. The transition function of each cell implements the image processing algorithm. There exist three types of functions that can be classified as *unitary*, *iterative* or *non- homogeneous.*

The unitary functions implement algorithms that utilize punctual or local operators and execute one iteration to obtain the result. The punctual operators have effect only on the pixel contained in the cell, while the local operators have effect on the cell neighbourhood. The *enhancement* algorithm that allows a best visual interpretation of an image is an example of punctual operator. *Filtering* and *convolution* algorithms are examples of local operators. Similarly, the extraction of a contour from a binary image on the basis of the presence of a 0-element in the 8-neighbourhood of a pixel also represents an algorithm of this class.

An iterative function implements algorithms that use the same function containing local operators in a loop until a termination condition is reached. *Thinning* algorithms or the *shift* of an image towards the right are examples of functions that require several steps for the execution.

When the parts of an image must be processed in different way the CA is *non-homogeneous.* In this case we do not have a single transition function for all cells but different functions that change in space and time.

With CAMEL we tested different algorithms belonging to the described classes. In all cases the writing of the transition function has required about one hour and the parallel execution of the algorithms has been always scalable. From our experience we can deduce that CAMEL allows the interactive development of parallel software for image processing without the programmer knowing parallel software implementation techniques.

4.6 Genetic algorithms

Genetic algorithms (GA) [11] are a class of adaptive general-purpose search techniques inspired by natural evolution. A GA works on a *population* of elements (chromosomes), generally encoded as bit strings, representing a sample, randomly chosen, of candidate solutions to a given problem. A GA is an iterative procedure that maintains a constant size population of individuals. Each member of the population is evaluated and a measure of its fitness as a solution is assigned to it. In each step a new population is generated by selecting the members that have the best fitness. To explore all the search space is necessary to alter the selected elements by means of *genetic operators* to form new elements to be evaluated.

The most widespread genetic operators are: the *crossover* and *mutation* operator. Crossover combines two strings to produce news strings with bits from both. Mutation randomly alters the bit at each string position if a probability test is passed. Crossover produces new search points. Mutation ensures that every area of the search space can theoretically be explored.

GA are inherently parallel and their parallel implementation allows to speed-up their execution. There are two main approaches for the parallel implementation of GA, the *migration* model and the *diffusion* model. The former divides the population in subpopulations and on each partition works a sequential GA. Genetic information is exchanged with low frequency by a random migration of solutions between subpopulations. In the diffusion model each chromosome has a spatial location and interacts only within some neighborhood. Clusters of solutions tend to form around different optima, which is both inherently useful and helps to avoid premature convergence. Information slowly diffuses across the grid.

The diffusion model allows a parallel implementation of GA using the cellular automata paradigm according to a *fine-grained* approach. A chromosome belonging to the initial selected population and its fitness are associated to each cell of a CA. The transition function associated with a cell provides, at each generation, to select the chromosome with the best fitness in the neighbourhood. Then the genetic operators of crossover and mutation are applied to the current string and the selected string. After evaluating the *offspring*, if one of them has a better fitness than the current string, it becomes the current string.

This approach has the advantage to work with large populations and allows to obtain a fastest convergence reducing the number of iterations and reducing the execution time. Changes to this approach, through the introduction of a migration step, are proposed by Tomassini [12] and implemented on a CM-2.

We have implemented by CAMEL the diffusion model, with and without the migration step, applying it to a global function minimization problem. The used function is defined as follows

$$f(x) = \sum_{i=1}^{10} x_i^2 / 4000 - \prod_{i=1}^{10} \cos(x_i / \sqrt{i}) + 1 , \qquad x_i \in [-600, 600]$$

In finding the global minimum of the function, the success percentage of models with and without the migration step has been similar to that one obtained by Tomassini, respectively 15% and 100%. Further, CAMEL allowed to set interactively the probability associated to the mutation operator and showed the result in a graphical way associating a colour to each chromosome.

5 Related work and conclusions

Rice et al. [13] defined a problem-solving environment (PSE) as "*a computer system that provides all the computational facilities necessary to solve a target class of problems. ... Moreover, PSEs use the language of the target class of problems, so user can run them without specialized knowledge of the underlying computer hardware or software. By exploiting modern technologies such as interactive color graphics, powerful processors ...*".

According that definition and on the basis of the the experiences discussed in the paper, CAMEL can be considered a problem-solving environment for computational modeling and simulation. In fact, CAMEL covers many user requirements and several important features can be highlighted from different viewpoints:

- CAMEL allows the implementation of fine-grained parallelism on a multicomputer system in a very efficient way by means of the SPMD model. Moreover, it provides animated visualization of the output of simulations.
- The CAMEL approach assumes a user with a little programming background sufficient to roughly understand the CAMEL philosophy, to quickly learn the CAMEL utilities and the programming interface necessary to define the transition function. In such a way, the CAMEL architecture becomes very transparent to a user, which benefits from a greater flexibility and deeper interaction features.
- The flexibility of the environment permits to balance a larger size of cellular space increasing the number of processors.
- There is no practical limit to the number of states of a cell so it is possible to reduce every other possible neighborhood to the directly available neighborhoods by a trade off with the

states. Furthermore, when a model must be validated, it is important to experiment different levels of detail.

- The MIMD architecture permits, by load balancing, to introduce time optimization for stationary regions, if any exists.
- The communication cost for the implementation is a negligible part (lower than 5%) of the total computation time.

CAMEL partially shares some of these features with other CA-based systems such as CAM [6], CAPE 7], and StarLogo [8]. On the other hand, several differences exist between CAMEL and these CA-based tools. For example, CAM-6, a specialized machine for CA, constitutes an engine that is difficult to surpass when a CA model can be defined in a simple way, however it does not allow, differently from CAMEL, the implementation of CA with a large number of states. In fact, CAM-6 is well suited for CA with no more than sixteen states and with a combination of the few neighborhood indexes. CAPE is a CA environment implemented on a transputer network. A first difference is given by the lack in CAPE of a load balancing strategy. This do not allow CAPE to optimize the excution times in case of stationary regions. Further, the rationale of CAPE is centered on a quick export of Fortran programs running on a sequential computer to a parallel machine. Finally, StarLogo is a CA-based programming environment designed by Resnick at MIT especially for educational use. The system has been implemented on a Connection Machine. The StarLogo programming approach is similar to the CAMEL one. Both systems offer a programming language for the difinition of the evolution rules of the lattice cells and allow a user to observe the global behaviour of the simulation on a graphical screen. The main differences from CAMEL concern the use of a SIMD computer for the StarLogo implementation and its utilization for educational purposes only.

In the paper we showed how CAMEL can be used as a versatile and scalable system that allows a user to solve complex problems exploiting the computing power offered by a highly parallel computer in a transparent way. The applications described here are an interesting example of use of a high-performance system to solve complex problems by an interdisciplinary collaboration among scientists and professionals coming from different fields. In the application design and implementation each one contributed with his expertise to define, model, implement, and evaluate the solutions to real problems developed by CAMEL.

Acknowledgements

This research is supported by CEC ESPRIT P.C.I. contract n° 94529419370.

We thank Mario Cannataro, Gino Mirocle Crisci, Demetrio Festa, Domenico Gattuso, Andrea Guglielmelli, Fiore Nicoletta, Marino Sorriso-Valvo, and Antonio Vitetta, who contributed to the design of applications of CAMEL discussed here. The authors would like to thank all reviewiers for their constructive comments and suggestions.

References

1. T. Toffoli, "Cellular Automata as an Alternative to (Rather than an Approximation of) Differential Equations in Modeling Physics", *Physica 10D,* 1984, pp. 117-127.
2. S. Wolfram, "Computation Theory of Cellular Automata", *Comm. Math. Phys.* Vol. 96, 1984, pp. 15-57.
3. J. von Neumann, *Theory of Self Reproducing Automata*, University of Illinois Press, 1966.
4. B. P. Brinch Hansen, "Parallel Cellular Automata: a Model for Computational Science", *Concurrency: Practice and Experience*, Wiley, Vol. 5, 1993, pp. 425-448.
5. Y. Feldman and E. Shapiro, "Spatial Machines: a More Realistic Approach to Parallel Computation", *Communications of the ACM*, Vol. 35, 1992, pp. 61-73.
6. T. Toffoli and N. Margolus, *Cellular Automata Machines A New Enviroment for Modeling*, The MIT Press, 1986.
7. M.G. Norman, J.R. Henderson, G. Main, and D. J. Wallace, "The Use of the CAPE Environment in the Simulation of Rock Fracturing", *Concurrency: Practice and Experience*, Wiley, Vol. 3, 1991, pp. 687-698.
8. M. Resnick, *Turtles, Termites, and Traffic Jams*, The MIT Press, Cambridge, Mass., 1994.
9. 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,* North-Holland, Vol. 21, April 1995.
10. S. Di Gregorio, D.C. Festa, R. Rongo, W. Spataro, G. Spezzano, and D. Talia, "A Microscopic Freeway Traffic Simulator on a Highly Parallel System", *Proceedings of ParCo'95,* Gent, Belgium, September, 1995.
11. J.H. Holland, *Adaptation in Natural and Artificial Systems*, *The MIT Press*, Cambridge, Mass., 1992.
12. M. Tomassini, "The Parallel Genetic Cellular Automata: Application to Global Function Optimization", *Proc. of the Int. Conf. on Artificial Neural Networks and Genetic Algorithms,* Springer-Verlag, Wien, 1993.
13. E. Gallopoulus, E. Houstis, and J. Rice, "Problem-Solving Environments for Computational Science, *IEEE Computational Science & Engineering*, Vol. 1, No. 2, 1994, pp. 11-23.