PARALLEL DISCRETE EVENT SIMULATION OF COMPLEX PROPAGATION PHENOMENA : PARALLEL INTERFACES AND CELLULAR ENVIRONMENTS

Eric INNOCENTI[*]
Alexandre Muzy
Antoine Aïello
Jean François Santucci
UMR CNRS 6134
U.F.R. Sciences et Techniques
University of Corsica
ino@univ-corse.fr

Key words
Parallel discrete event simulation, cellular automata, parallel interfaces, fire modelling and simulation.

Abstract

In this paper, we study principles and techniques of Parallel Discrete Event Simulation that are used for the simulation of environmental complex phenomena on parallel computers. The rise of new concepts in this domain allows to develop tools for the study of such systems. Among them, the paradigm of cellular automata facilitates the building of spatial models and the integration of parallel techniques. Some parallel programming interfaces are developed to accelerate the simulation treatments and exploit the computing ressources provided by the parallel computers. The Combination of the cellular automata and the parallel interfaces makes it possible to solve the modelling and simulation problems arisen by the complex systems considered. We present the modelling tools available for the expression of the space dynamics of the complex phenomenon considered and how to extend the possibilities of the traditional models for spatial modelling and simulation.

Introduction

Studying global ecosystem is nowadays essential to understand the human impact on such a fragile equilibrium. The modelling of environmental complex phenomena which occur in constitutes a necessary starting point. In fact, precise models only permit to design tools for a better understanding and to facilitate decision making. With the computing development, the simulation is becoming the best methodological choice possible to supply numerous solutions. However, the simulation of complex environmental phenomena often implies spatial effects which generate major problems in terms of memory and time computing. In addition the tools developed must predict the behavior of the phenomena in advance in order to operate the best way. Thus, as far as the simulation of forest fire propagation is concerned the development of a simulation environment requires the use of techniques able to apprehend the difficulties arisen by these issues. Parallel simulation has recently become an essential tool. Combined with the techniques of discrete events simulation and with the cellular automata paradigm, it makes it possible to solve the difficult problems related to the programming of such models. The parallelisation of these models depends on parallel programming interfaces.

This document deals with the use of parallel interfaces and cellular automata in discrete event simulation. Thanks to this combination, it is possible to solve the problems aroused by the simulation of complex propagation phenomena such as vegetation fires. A first part presents the paradigms of the Parallel Discrete Event Simulation. A second part deals with the principal interfaces for parallel programming. A third part presents some environments which exploit parallelism based on cellular models. Eventually, a discussion starts the reflexion which lays the bases of the research tasks we plan to develop in order to model the propagation of vegetation fires.

[*] Corresponding author

1 Paradigms of discrete event simulation

Discrete event simulation implements models whose state variables evolve in time in a discrete way. Contrary to continuous simulations, it is advisable to distinguish at what time the changes of states occur on the time axis. The events of the simulation correspond at these changes. The management of these events is executed in the timestamp order of their occurrences. Two principal methods are used to manage the simulation time: one driven by the clock, the other driven by the events. These two methods are at the origin of two techniques of discrete events simulation.

1.1 Discrete time and discrete event model

The simulation of dynamic systems is often intuitively developed using discrete time models [Zeigler and Al, 2000]. The formalisation of such models supposes a step by step execution of the simulation. This kind of simulation technique is used to develop models where the time of the simulation evolves using fixed time steps. The time axis is delimited in identical intervals. A unique clock gives rhythm to the different components of the model. At each time step, the simulator executes the events occurring between the current time and the next clock signal. Discrete time simulation is illustrated figure 1.

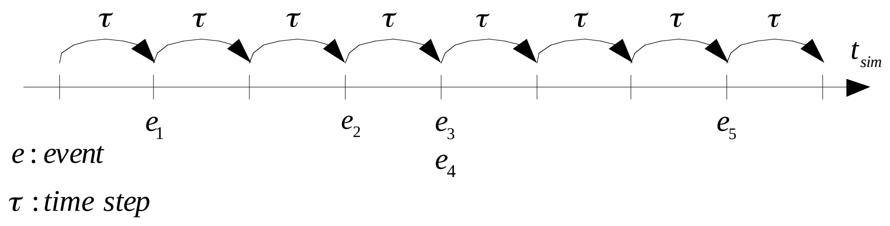

Figure 1. *Discrete time simulation principle.*

Discrete time simulations describe step by step the evolution in time of real systems based on discrete models. The most intuitive discrete time models to simulate spatial dynamic systems are founded on cellular automata paradigm.
The event oriented simulation implies the development of models where the simulation time progresses by leaps of time occurrences [Erard and Déguénon, 1999]. In this case, only the significant instants where the model evolves are taken into account. It is necessary to develop a scheduler in order to manage chronologically the events of the simulation. Each time step implies a treatment. The time of the simulation progresses according to the events programmed in the scheduler. The treatment of an event at the head of the scheduler can generate new additions or suppressions of other events. The simulation directed by the events is illustrated figure 2.

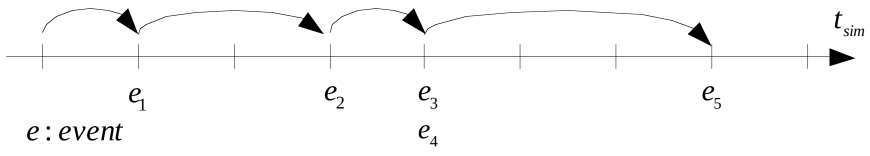

Figure 2. *Oriented events simulation principle.*

This kind of approach is used with the DEVS specification for discrete models [Zeigler, 1976]. Parallel discrete event simulation (PDES), refers to the execution of a single discrete event simulation program on a parallel computer [Fujimoto, 1990]. PDES requires the understanding of algorithmic bases which it is necessary to introduce.

1.2 Parallel Discrete Event Simulation

Parallel Discrete Events Simulation (PDES) deals with the simulation of the discrete models presented previously on a parallel architecture. In the case of parallel simulations, the various processes generated by the model share a common memory or shared memory which enables them to exchange information. It is necessary in this case to establish mechanisms of management in order to avoid any access conflict to the memory. This type of algorithms is usually implemented on shared memory multiprocessors. In the case of distributed simulations, the model is broken down into logical processes (LPs) which require mechanisms for exchanging information across a network of workstations. The memory is individual and the communications are based on messages passing techniques, whose MPI is the standard [MPI, 1995, Gropp and Al, 1999]. It is implemented in the manufacturers libraries and in the free software libraries such as MPICH [Gropp and Lusk., 2003] and LAMMPI [Burns and Al, 1994]. Of course, parallel simulation and distributed simulation can cohabit within the same simulation environment; the two types of PDES are not exclusive. The literature declines the algorithms of the PDES in two categories: synchronous parallel algorithms and asynchronous parallel algorithms. The difficulties which result from the synchronization of the logical processes tasks constitute the heart of the PDES problems.

1.2.1 Synchronous parallel simulations

The logical processes of synchronous parallel simulations are executed at the same virtual time, their local clocks progress on the basis of a common virtual time. This central management of time is executed using a global clock whose implementation can be centralized [Nutaro, 2000], or distributed [Ferscha and Tripathi, 1994]. If the time is centralized the clock is implemented using an algorithmic structure which is common to the different local processes. If the time is not centralized, it is distributed in each process of the simulation. Then, an algorithmic structure maintains a local representation of the global clock. All along the simulation, the virtual time is unique and identical for all the processes of the network. Such algorithms require the installation of synchronization barriers, i.e. algorithmic mechanisms for processes synchronization executed at each occurrence of events. [Fujimoto, 1990] recalls that the development of such simulators on shared memory multiprocessor limits the costs of interprocess communications. As we said previously, the synchronous algorithms can be driven by time or by events. The synchronous PDES concepts are summed up in figure 3.

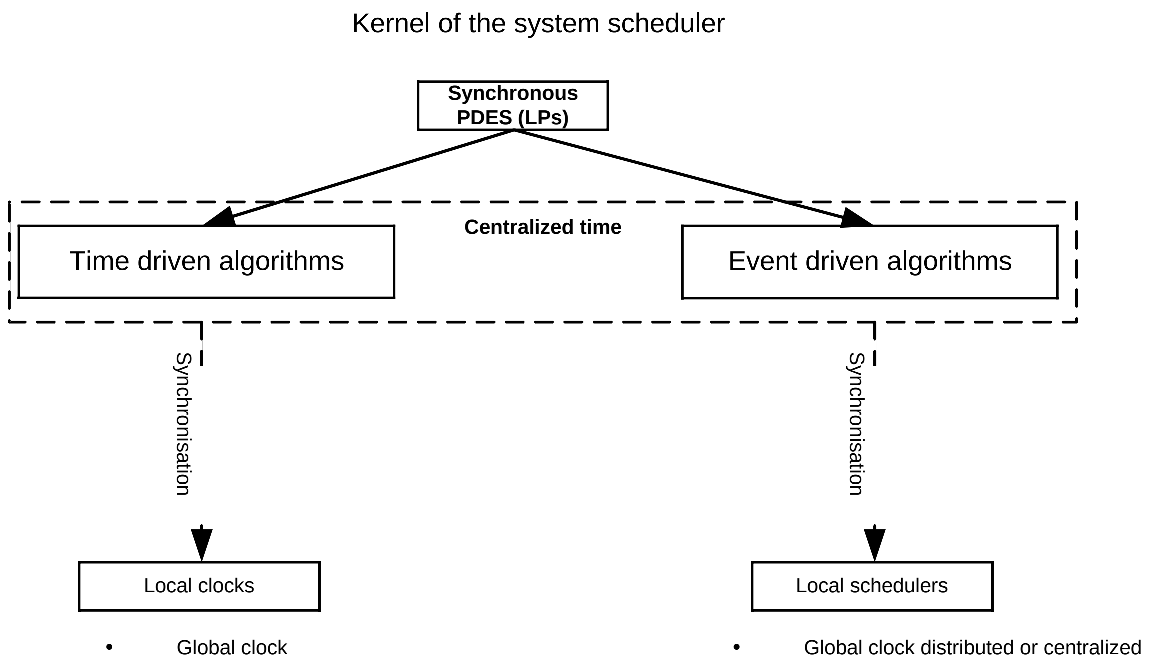

Figure 3. *Synchronous Parallel Discrete Event Simulation.*

1.2.2 Asynchronous parallel simulations

In asynchronous simulations LPs are not synchronized starting from a global clock. The simulation time is managed independently in each process with synchronizations only occurring few times during the simulation process. These synchronizations require the installation of a simulator in each process [Ingels and Raynal, 1989]. The latter doesn't know the global state of the system, and advance the simulation time using the local information and the contents of the messages that it receives from the other processes. The major problem encountered is the requirement for each logical process, to execute the treatments associated to the messages in a strict temporal order, to certify the exactitude of the simulation results. This condition is called by R.M. Fujimoto "causality constraint" which can be summed up as follows: "the future cannot influence the past" [Fujimoto, 1990]. This implies processes exchanging time stamped messages, i.e. messages including temporal information associated to the treatments. The messages received by the processes are then consumed in a time stamped order, which implies the respect of the causality constraint at the local level. In order to guarantee this one, different simulators were proposed, and the literature classifies them into two categories: conservative asynchronous simulators and optimistic asynchronous simulators.

In the case of a conservative asynchronous simulator, a logical process runs the treatment of a message with stamp 't_1', only if it is certain that no other message with stamp 't_2', such as '$t_2 < t_1$' can reach it. However, when a process is waiting for message, this can generate a dead lock situation [Misra, 1986]. Two techniques are then used. The first technique consists in detecting dead lock; the simulator executes a treatment able to restore a safe situation; a second technique consists in preventing strictly dead lock situations, using null-messages. The latter are used to indicate to the other processes that no effective message will be sent. This method prevents dead lock situations, but can lead to an undesirable proliferation of null-messages. The conservative algorithms were developed initially by [Chandy and Misra, 1979], since numerous evolutions and optimizations like [Lemeire and Al, 2004] and [Bui and Al, 2003] were proposed.

Optimistic asynchronous simulator doesn't take care of causality constraint. If a message arrives in a process with a time stamp 't_2' lower than that of the last event considered 't_1' such as '$t_2 < t_{1'}$, then the simulator will cancel all the events executed before 't_1'; this is the "rollback procedure". Moreover, the process must send antimessages to its attached processes in order to tell them to cancel operations posterior to 't_2'. An optimistic approach requires recording preceding states before the validation of the current state by the principal process: this is the "fossil collection". The fossil collection is calculated from the "Global Virtual Time", which represents the temporal limit until which it is possible to execute a rollback procedure. Records of states and memory needs are the major disadvantages of optimistic simulations.

2 Parallel programming interfaces

Many interfaces make it possible to abstract parallelism more or less and each of them offers a compromise between comfort of programming, portability and efficiency. The literature essentially separates them into two categories: programming libraries and environments. The libraries, contrary to environments, are often presented as extension of usual programming languages like C, C++, Java, FORTRAN, etc..., with which they associate new functions for parallel programming. Another solution consists in writing sequential code, the parallelisation of the tasks remaining in charge of the compiler, but this document doesn't deal with it. In this section the most representative parallel programming interfaces are presented. For each of them, essential features are presented and the advantages and drawbacks related to their use.

2.1 Parallel Virtual Machine

PVM (Parallel Virtual Machine) is a software package from which it is possible to conceive virtually a parallel calculator from a heterogeneous and geographically distributed computer network [Geist and Al, 1994]. Heterogeneity can relate to the architecture of the processors and to the operating systems. PVM appears as a library of communications making it possible to exchange messages between the processes of an application. The user defines a set of machines which is interpreted like a distributed memory multiprocessor machine. The term of "parallel machine" refers to this virtual multiprocessor machine, whereas the term of "host" refers to one of the machines that is member of the network. The communications between the hosts are done with PVM which is responsible for converting the data between two machines that don’t have the same internal representations of data. PVM is also conceived to be able to work with different kinds of networks. PVM fulfils all its functions thanks to daemons which it installs when it starts on each host. The daemons manage the communications between the hosts. Before executing a PVM program, it is necessary to define the set of hosts and to start PVM. When this operation is executed, it is then possible to launch a PVM program. The environment is managed using options through a PVM console on the host machine. The advantages and the disadvantages of PVM are summarized in table 1.

PVM	
Advantages	**Drawbacks**
The addition of a set of new machines is carried out in a transparent way for the virtual machine;	The parallel programming is explicit, the management of the communications and synchronizations is entirely in charge of the developer;
There is a public domain version of PVM	Development cost
PVM work on homogeneous as well as on heterogeneous computer networks;	The programs are more difficult to read (division and distribution of the application);
The PVM program is portable on various configuration;	The implementation is often difficult;
PVM is easy to install.	PVM is founded on low level concepts.

Table 1. *PVM advantages and drawbacks.*

PVM initiated the definition of the MPI standard (Message Passing Interface). A comparison between PVM and MPI is done in [Gropp and Al, 2002]. It should be noticed that PVM is effective for heterogeneous networks of machines, and it is not the case of MPI-1 (cf. following section).

2.2 Message Passing Interface

MPI (Passing Message Interfaces) is a specification for libraries of messages passing which was developed in the early 1990s and which aims at giving an answer to the need of clarification in the field of parallel computing [MPI, 1995, Gropp and Al, 1999]. Initially, parallel machines were came with their own communication library which generated important problems of evolutionary and portability. The update of the programs was extremely difficult and required a lot of work. Academics and industrials from all scientific domains thus ended up getting along with the MPI standard. An MPI library provides the functions of communication for the different nodes of the distributed memory architecture. The users of such a library can use programs developed in C, C++, and FORTRAN, which are compiled using the traditional compilers, but linked to the corresponding MPI libraries. An MPI library can be used on parallel machines and on workstations clusters (heterogeneous from MPI-2). The advantages and the disadvantages of PVM are summarized in table 2.

MPI	
Advantages	**Drawbacks**
There are many libraries and codes;	MPI requires to cut the simulation model, which is often difficult to realize;
MPI is portable onto many architectures;	It is essential to reduce the communications in order to obtain better performances;
The user decides parallelization (load balancing, distribution of calculations and data);	The implementation is complex;
Very good flexibility.	The modifications of the sequential code are impossible without important modifications;
	MPI is based on low level concepts

Table 2. *MPI advantages and drawbacks.*

A new MPI-2 standard was developed few years ago, however very few libraries implement it. This new standard gives the functionalities of MPI libraries which are closer to and more stable than the PVM one. Consequently, it is rather preferable to develop with MPI than with PVM [Fadlallah and Al, 2000].

2.3 Portable Operating System Interface

POSIX (Portable Operating System Interfaces) is a standard providing the mechanisms to create processes, the programmer must take care of the synchronization between these processes by using low level primitives, such as lock or condition variables [POSIX, 1996]. POSIX threads are usually called "pthreads". The pthread library is an implementation of the POSIX 1003.1c standard. It provides primitives to create activities (or light processes) and to synchronize them. These primitives are similar to those provided by the other libraries (Solaris LWP, Windows) or languages (Modulated-3, Java). In the domain of the light processes, the POSIX standard is essential, more precisely on Unix OS. The advantages and the drawbacks of the threads based on the POSIX standard are summarized in table 3.

POSIX	
Advantages	**Drawbacks**
The load balancing is transparent for the user	MOSIX depends on the Linux kernel
Advantages of a free software	Applications using shared memory or light weight processes can not migrate in the cluster.
User interventions are limited	
MOSIX can run on heterogeneous systems	
MOSIX does not imply the modifications of the applications	
MOSIX is able to manage dynamically the ressources on the different nodes	

Table 3. *POSIX advantages and drawbacks.*

2.4 OpenMP

OpenMP is a portable standard which makes it possible to parallelize the code of the simulation models on homogeneous architectures with shared memory (SMP) [OpenMP, 2002]. Thanks to a set of directives it is possible to describe the parallelism simply. This set of directives is based on the concept of parallel loops and shared or private variables. The goal of the OpenMP standard is to increase the portability of the parallel programs intended for shared memory architectures. The specifications of OpenMP are decided by "OpenMP Architecture Review Board". The execution of the parallel loops of OpenMP is based on the fork-join programming model. In a parallel section the thread running is divided into groups of threads which, will be synchronized at the end of the section and will join into only one thread which is, the first original thread. OpenMP makes it possible to easily specify the sharing and synchronization of tasks, by providing directives which have the same syntax. It should be noticed that for the moment, OpenMP does not propose directives which make it possible to distribute the data (clustering). However, with the considerable increase of the number of SMP architectures in high performance clusters (HPC), the current tendency consists in developing hybrid solutions openMP/MPI. The nodes of the cluster are SMP machines whose loops of iteration use openMP, and the communications between the nodes are done using MPI [He and Ding, 2002, Henty, 2000]. The advantages and the disadvantages of the openMP standard are summarized in table 4.

OpenMP	
Advantages	**Drawbacks**
Simplicity of use, fast installation	No coarse grain parallelism;
No explicit communication between processes, high level concepts	Fine grain parallelism not efficient when the processes are numerous
Portability on shared memory architecture (SMP)	Very few libraries available
Easy transition between sequential and parallel code	Sensitive to the load of the machine if communications are numerous
Efficient on SMP architecture	Limited control of the distribution of the data and calculation
High level parallelization	

Table 4. *OpenMP advantages and drawbacks.*

3. Cellular parallel models

In the literature there are many environments which make it possible to specify cellular models. [Jorba and Al, 2002] define this kind of models on two levels of specification: local and global levels of components. Thus, the spreading model is described on two different scales of abstraction. At a high level, the propagation phenomenon is considered as a whole unit which evolves in time and space. At the local levels, elements of smaller size (cells) are modelled. Those local models take into account particular conditions of each element and their neighbours, in order to compute their evolution in the time. If we rest on this modelling approach, the propagation models conceived are often complex and need a lot of computing resources. But, they must be simulated in a sufficiently short time, so that decision making are possible. Parallel Discrete Event Simulation (PDES) can give the computing power needed for simulating this kind of models. The latter is implemented through environments of simulation in order to evaluate the performances of the models in different conditions. In this section we present four simulation environments for propagation phenomena taken from the literature. Although this list is not exhaustive, it allows us to analyse the principal features which support the design of such tools.

3.1 CD++ environment

[Ameghino and Al, 2003], apprehend the complexity of the real systems according to a microscopic approach, i.e. oriented to the description of the small entities. Then, the simulation models use a cellular automaton and transition rules in order to implement the simulation. This type of approach is used in CD++, a modelling and simulation environment which implements the Cell-DEVS formalism. The latter is an extension of the DEVS formalism which allows to specify cellular models. Complex systems are easily described with this tool. Cellular elements which constitute the propagation domain are defined with DEVS formalism and are described as atomic models. The original features are recently extended in order to integrate parallelisation techniques [Troccoli and Wainer, 2001]. Parallel Cell-DEVS has been defined to integrate Parallel-DEVS formalism. CD++ environment was modified consequently and allowed distributed simulation of Parallel Cell-DEVS models. Figure 4 is an example of model specification of heat diffusion with CD++.

```
[top]
components : surface Heat@Generator Cold@generator
link : out@Heat inputHeat@surface
link : out@Cold inputCold@surface

[surface]
type : cell
width : 100
height : 100
delay : transport
defaultDelayTime : 1000
border : wrapped
neighbors : surface(-1,-1) surface(-1,0) surface(-1,1)
neighbors : surface(-1,-1) surface(-1,0) surface(-1,1)
neighbors : surface(-1,-1) surface(-1,0) surface(-1,1)
initialvalue : 24
in : inputHeat inputCold
link : inputHeat in@surface(25,25)
…
localtransition : heat-rule
portIntransition : in@surface(25,25) setHeat
…
[heat-rule]
rule :{((0,0)+(-1,-1)+(-1,0)+(-1,1)+(0,-1)+(0,1)+(1,-1)+(1,0)+(1,1))/9}10000{t}
```

Figure 4. *Specification of a heat diffusion model using CD++.*

The fact that the environment rests on a formal specification for modelling the phenomenon facilitates the writing of propagation models. Moreover, a high level language is proposed with the environment, thus considerably reducing implementation times. The major inconvenient of the Cell-DEVS formalism is inherent to the nature of the cellular elements. In fact, the latter being implemented as atomic DEVS models, they need to exchange an important number of messages through the simulation phase. This constraint seems to be extremely penalizing in the case of the simulation of propagation phenomena such as fire spreading [Muzy et al., 2002a]. It seems that the recent quantum theory aims at reducing messages exchanges and then to enhance simulation times [Muzy and al., 2002b].

3.2 DEVS C++

[Zeigler and al., 1996] rest on the principles of the object programming in order to build modular representations of complex phenomena. DEVS C++ is a simulation environment resting on the DEVS formalism and developed at Arizona University. It allows to analyse, conceive and simulate discrete event systems. DEVS-C++ environment is developed from an architecture which uses the C++ language for programming models and simulators. Simulation architecture rests on the Object Oriented Programming, which enables to easily adapt the execution on sequential or parallel platforms. For that, the DEVS formalism is explained within classes which interact and hide the details of the implementation (sequential or parallel). The final user uses the objects interfaces in order to specify the model he wants. The DEVS models are built with the help of classes named container. This approach facilitates the definition of models and their manipulations. This approach is illustrated in figure 5.

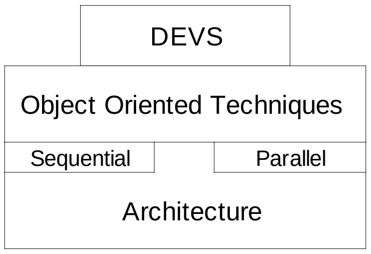

Figure 5. *Object oriented implementation of the DEVS model in DEVS-C++.*

In spite of a modular and extensible approach, this simulation environment doesn't allow to exploit efficiently cellular models, more precisely in the case of the simulation of complex propagation phenomena. In fact, as for CD++, the cells of the domain are represented as atomic DEVS classical models, which considerably limit the performances of the simulation [Muzy and al., 2002a].

3.3 CAMEL
[Cannataro and al., 1995] propose the CAMEL environment, which allows a fine setting of the simulation model in order to simulate many kinds of complex propagation phenomena, as lava floods, car traffic, landslides…. Recently, CAMEL has been used in the project CABOTO (Cellular Automata for the Bioremoval of Toxic Contaminants) [Spezzano and al., 1998]. The main objectives of this project deals with the use of cellular automata in order to model and simulate the reprocessing of contaminated soils. More particularly, such models describe the processes of the reprocessing which could be realised in simulating the growth of the bacteria. These models allow to predict the operation of reprocessing on large scale, from the knowledge of geological data, chemical, and microbiological, and from the results of experimentations. The cellular model used for the simulation is three dimensional, it is conceived for the large scale simulations, and it can describe heterogeneous domains. The neighbour of each cell is composed of six other cells, four of them being at the same height (north, south, east, west), whereas the two remaining cells are in the high and low positions. The neighbour of each cell is described figure 6.

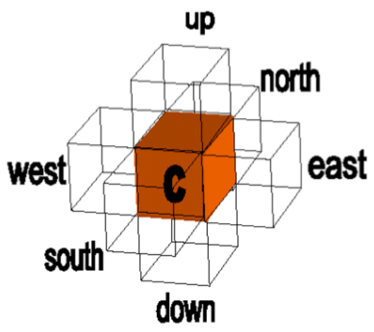

Figure 6. *Tridimensional neighbour of a cell (c) in the CAMEL environment.*

The model has a structure in layer where the first layer describes the flow of the fluid through the ground, the second layer describes the behavior of the chemicals (dissolved or adsorbed on the surface), and the third layer describes the interactions between the chemicals and the biomass. The results of the execution show that the simulation of the whole of the events of decontamination (which require 64 days complete for a grid 128X15X11, 21120 cells), are obtained in 19 days of simulation using a monoprocessor machine, whereas, only approximately 16 hours are necessary using a 32 multiprocessors machine. These results underline the interest to use parallel computing for simulating the simulation of the propagation of complex phenomena and the effectiveness which results from the cellular implementations which exploit parallelism. CARPET is the programming tool from which it is possible to define the cellular models in CAMEL. Its programming language is similar to the C programming language and is used to describe the cells of the automaton.

3.4 StarLogo
[Resnick, 1997] combines cellular automata and agents. The in fine objective of this modelling is the micro simulation through a multi agent approach of the dynamic systems on a large scale where the analysis requires the comprehension and the modelisation of individual and collective behaviours. StraLogo rests on an extension of the programming language Logo conceived at the MIT. It's a multi-agent language giving simple constructions which allow to define the evolution rules of the cells composing a cellular automaton. It is conceived in Java (Object Oriented Language), and it is portable onto any platform.

Moreover, it allows a non computer scientist to develop programs without knowledge of the classical programming languages. It is relatively simple to handle and it has integrated tools which facilitate the treatment of the results. An example of application is the forest fire spreading. In this case the StarLogo environment makes it possible to change the density of the trees and reiterates simulation under various conditions, in order to carry out the different possible evolutions of the fire propagation. The users then observe the global behavior of all the cells on a graphic screen, as represented in figure 7.

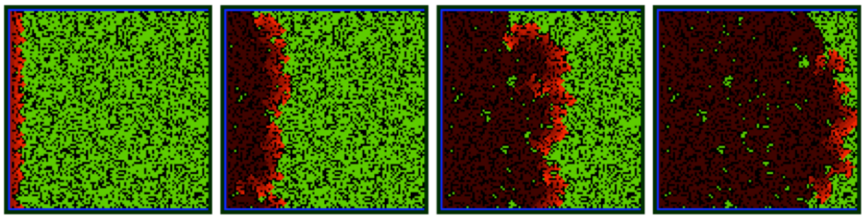

Figure 7. *Four snapshots of the simulation of fire spreading with StarLogo.*

The thousands of graphical elements are executed in parallel and are programmed to react to their environment. The general principle of this type of model is to make a population of reactive agents evolve and to observe the result emerging after a number of program iterations on a screen. The simulations are described in a virtual world using three basic elements: the turtles (active agents), the patches (passive objects of the environment), and the observer which can send patches to the turtles and the patches. Thus, the world consists of agents which obey instructions. Each agent interacts with the world, and this parallel to the other agents placed in the environment. The turtles move in the world which is two-dimensional and divided into a grid of patches. Each patch is a square on which a turtle can evolve. The observer is not located. At the beginning of the simulation there are no turtles and the observer has the possibility of creating turtles and patches. NetLogo is a distributed version of Starlogo [NetLogo, 1999].

4. Conclusion and perspectives

Theory of modelling and simulation and its parallel extension provides the methods and techniques to use for modelling complex propagation phenomena. Modelling such systems is facilitated by the use of the cellular models. In fact, they make it possible to express the space dynamics of the phenomena in the time and to apprehend progressively the complexity. They extend the possibilities of the traditional models of cellular automata and make it possible to simulate a greater number of complex phenomena. It is possible to associate to them the techniques of parallel discrete events simulation, which make it easier concurrent execution of the treatments of the simulation. Formal methods of modeling help in developing and are used through simulation environments where rules and architectures appreciably improve the process of development in terms of time, software legibility, evolution, and portability. Moreover, the parallel architectures offer to these environments the suitable, evolutionary and powerful physical support. The study of some environments of simulations made it possible to underline important concepts which initiate the reflexion from which we try to give an answer to the problems arisen by the simulation of complex propagation phenomena.

References

[Ameghino and Al, 2003] Ameghino, J., Glinsky, E., Wainer, G. (2003). Applying Cell-DEVS in models of complex systems. Dans *Proceedings of the 2003 Summer Computer Simulation Conference. Montreal, QC, Canada*.

[Bui and Al, 2003] Bui, P., T., Lang, S., D., Workman, D., A. (2003). A New Conservative Synchronization Protocol for Dynamic Wargame Simulation. Dans *Proceedings of 2003 Spring Simulation Interoperability Workshop*.

[Burns et al., 1994] Burns, G., Daoud, R., Vaigl, J. (1994). LAM: An open cluster environment for MPI. *In Proceedings of Supercomputing Symposium'94 (J.W. Ross, ed.), , University of Toronto, pp 379–386*.

[Cannataro and al., 1995] Cannataro, M., Di Gregorio, S., Rongo, R., Spataro, W., Spezzano, G., Talia, D. (1995). A Parallel Cellular Automata environment on Multicomputers for Computational *Science. Parallel computing 21*, pp. 803.

[Chandy and Misra, 1979] Chandy, K., M., Misra, J. (1979). Distributed Simulation : A case Study in Design and Verification of Distributed Programs. Dans *Proceedings of IEEE Transaction on Software Engineering SE-5 (5),* pp. 440-452

[Erard and Déguénon, 1999] Erard, P.,J. Déguénon, P. (1999). Simulation par évènements discrets. Concepts et réalisations en Simula, Ada et Smalltalk. *Presses Polytechniques et Universitaires Romandes*.

[Fadlallah and Al, 2000] Fadlallah, G., Lavoie, M., Dessaint, L. A. (2000). Parallel Computing Environments and Methods. École de Technologie Supérieure. Dans *Proceedings of International Conference on Parallel Computing in Electrical Engineering (PARELEC'00)* August 27 - 30, Quebec, Canada, p. 2.

[Ferscha and Tripathi, 1994] Ferscha, A., Tripathi, S., K. (1994). Parallel and Distributed Simulation of Discrete Event Systems. *Technical Report Report CS-TR-3336, University of Maryland (Dept. of Computer Science)*.

[Fujimoto, 1990] R.M. Fujimoto. « Parallel Discrete event Simulation ». Communications of the ACM, 33(10) . pp. 30-53, October 1990.

[Geist and Al, 1994] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., Sunderam, V. (1994). PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing. *MIT Press, Scientific and Engineering Computation Janusz Kowalik, Editor*. En ligne : http://www.netlib.org/pvm3/book/pvm-book.html

[Gropp and Al, 1999] Gropp, W., Lusk, E., Skjellum, A. (1999). Using MPI - 2nd Edition: Portable Parallel Programming with the Message Passing Interface. *Scientific and Engineering Computation series. MIT Press; 2nd edition*.

[Gropp and Al, 2002] Gropp, W., Lusk, E. (2002). Goals Guiding Design: PVM and MPI. Argonne National Laboratory. *Dans proceedings IEEE International Conference on Cluster Computing (CLUSTER'02), September 23 - 26, Chicago, Illinois, pp. 257-265*.

[Gropp and Lusk., 2003] Gropp, W., Lusk, E. (2003). Installation and User's Guide to MPICH, a Portable Implementation of MPI. Version 1.2.5.

[He and Ding, 2002] He, Y., Ding, C., H., Q. (2002). MPI and OpenMP Paradigms on Cluster of SMP Architectures: the Vacancy Tracking Algorithm for Multi-Dimensional Array

Transposition. Dans *Proceedings of Conference on High Performance Networking and Computing. Proceedings of the 2002 ACM/IEEE conference on Supercomputing. Baltimore, Maryland, pp. 1 – 14*.

[Henty, 2000] Henty, D., S. (2000). Performance of Hybrid Message-Passing and Shared-Memory Parallelism for Discrete Element Modeling. Dans *Proceedings of Supercomputing 2000, November 4-10, Dallas, Texas, USA. IEEE Computer Society, CD-ROM, ISBN 0-7803-9802-5*.

[Ingels and Raynal, 1989] Ingels, P., Raynal, M. (1989). Simulation répartie de systèmes à évènements discrets : Partie 1, modélisation et schémas d'exécution. *Rapport de recherche, INRIA-Rennes*.

[Jorba and Al, 2002] Jorba, J., Margalef, T., Luque, E., Campos da Silva André, J., Viegas., D. X. (2002). Parallel Approach to the Simulation Of Forest Fire Propagation. Dans *Proceedings of Environmental Communication in the Information Society. 16th International Conference "Informatics for Environmental Protection". September 25-27, Vienna University of Technology*.

[Lemeire and Al, 2004] Lemeire, J., Brissinck, W., Dirkx, E. (2004). Lookahead Accumulation in Conservative Parallel Discrete Event Simulation. (2004). Dans *Proceedings of High Performance Computing & Simulation (HPC&S) Conference*.

[Misra, 1986] Misra, J. (1986). Distributed Discrete-Event simulation. Dans *Proceedings of Computing Surveys, Vol 18. No 1, pp. 39-65*.

[MPI, 1995] Message Passing Interface Forum. MPI : A Message-Passing Interface Standard. (1995). Technical report, University of Tennessee, Knoxville, TN, June 1995. Version 1.1

[Muzy and al., 2002a] Muzy, A., Wainer, G., Innocenti, E., Aïello, A., Santucci, J., F. (2002). Comparing simulation methods for fire spreading across a fuel bed. Dans *Proceedings of AIS'2002. Lisbon, Portugal.*

[Muzy and al., 2002b] Muzy, A., Wainer, G., Innocenti, E., Aïello, A., Santucci, J., F. (2002). Cell-DEVS quantization techniques in a Fire Spreading application. *Dans Proceedings of 2002 Winter Simulation Conference*. San Diego, CA. USA.

[NetLogo, 1999] (1999). NetLogo User Manual version 2.0.0. http://ccl.northwestern.edu/netlogo/docs/.

[Nutaro, 2000] Nutaro, J. (2000). *Time Management and Interoperability in Distributed Discrete Event Simulation*. Thèse de doctorat. Electrical and Computer Engineering Dept., University of Arizona.

[OpenMP, 2002] Official OpenMP Specifications. C/C++ version 2.0. (2002). http://www.openmp.org/.

[POSIX, 1996] (1996). Portable standards Committee of the IEEE. Information Technology – Portable Operating System Interface (POSIX) – Part 1 : System Application Program Interface (API), 1996-07-12 edition, 1996. ISO/IEC 9945-1, ANSI/IEEE Std. 1003.1.

[Resnick, 1997] Resnick, M. (1997). Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds (Complex Adaptive Systems). MIT Press.

[Spezzano and al., 1998] Spezzano, G., Talia, D. (1998). Designing Parallel Models of Soil Contamination by the CARPET Language. Future *Generation Computer Systems*, 13, pp. 291-302.

[Troccoli and Wainer, 2001] Troccoli, A., Wainer, G. (2001). Performance analysis of Cellular Models with parallel Cell-DEVS. *Dans Proceedings of 2001 Summer Computer Simulation Conference. Orlando, FL. USA*.

[Zeigler and Al, 2000] Zeigler, B., P., Praehofer, H., Kim, T., G.. (2000). Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems. *Academic Press*.

[Zeigler and al., 1996] B.P. Zeigler, Y. Moon, D.Kim, T. G. Kim. (1996). DEVS-C++ : A High Performance Modelling and Simulation Environment. *Dans proceedings of 29 th Haiwaï international Conference onsystem Sciences (HICSS'96). Volume 1 : Software technology and Architecture, January 03-06, pp.350-360*.

[Zeigler, 1976] Zeigler B., P., (1976). Theory of Modelling and Simulation. *Wiley, New York*.