Environmental Modelling & Software xxx (2009) 1–13

Contents lists available at ScienceDirect

Environmental Modelling & Software

journal homepage: www.elsevier.com/locate/envsoft

A software framework for fine grain parallelization of cellular models with OpenMP: Application to fire spread

Eric Innocenti [a], Xavier Silvani [a], Alexandre Muzy [a], David R.C. Hill [b,*]

[a] University of Corsica, SPE – UMR CNRS 6134, B.P. 52, Campus Grossetti 20250 Corti, France
[b] Blaise Pascal University, ISIMA/LIMOS UMR CNRS 6158, BP 10125, Campus des Cézeaux 63177Aubière Cedex, France

ARTICLE INFO

Article history:
Received 10 November 2006
Received in revised form
13 November 2008
Accepted 18 November 2008
Available online xxx

Keywords:
Software framework
Fire spread physical model
DEVS
Open MultiProcessing (OpenMP)
Symmetric multiprocessors (SMP)

ABSTRACT

We are dealing here with the parallelization of fire spreading simulations following detailed physical experiments. The proposal presented in this paper has been tested and evaluated in collaboration with physicists to meet their requirements in terms of both performance and precision. For this purpose, an object-oriented framework using two abstraction levels has been developed. A first level considers the simulation as a global phenomenon which evolves in space and time. A local level describes the phenomena occurring on elementary parts of the domain. In order to develop an extensible and modular architecture, the cellular automata paradigm, the DEVS discrete event system formalism and design patterns have been used. Simulation treatments are limited to a set of active elements to improve execution times. A new kind of model, called Active-DEVS is then specified. The model is computed with a fine grain parallelization very efficient for present day multi-core processors which are elementary units of modern computing clusters and computing grids. In this paper, the parallelization with Open MultiProcessing (OpenMP) standard directives on Symmetric MultiProcessing (SMP) architectures is discussed and the efficiency of the retained solution is studied.

Software availability

Name of software: The C++ code is freely available with a simple request to Eric Innocenti ino@univ-corse.fr.

1. Introduction

Simulation is a powerful tool that enables a better understanding of real world problems. Among the various simulation techniques, the simulation of discretized differential equations has been used to describe complex systems and interpret many real world problems. To deal with space, these simulation models are usually represented as cell spaces (Wu et al., 2004; Langlois and Phipps, 1997; Karafyllidis and Thanailakis, 1997).

Fire spreading is a good example of complex phenomena that relies on differential equations and cellular models. In fire spread modeling, discrete equations are written down from the conservative laws for mass, momentum and energy that govern the system. In this paper, the domain context is multi-scale, from small-scale laboratory experiments to field scale experiments, hence, the physical complexity of the thermodynamic system under consideration strongly increases. In actual fires, a set of new control parameters and distributions appear in comparison with laboratory fires. These are mainly governed by a multi-scale structured vegetation (shrub or forest), the turbulent flow regime of the flame front and its crossing wind flow and the fractal nature of the topography. This enlargement in the range of scales (e.g. from few centimeters of a leave to several kilometers) supposes huge computing resources in order to capture each representative scale governing the conservative laws. The numerical resolution of the partial differential equations forming the system of conservative laws for a reactive fluid flow is known as Computational Fluid Dynamics (CFD), including multiphase flow simulations (Morvan and Dupuy, 2004) and Large Eddy simulations (Mell et al., 2007).

An efficient alternative to CFD for fire simulation is the cellular automata approach. Indeed, cellular automata appear to be appropriate for modeling such complex spatial phenomena as large-scale fires, because of their discrete nature and their suitability for implementation on computers (Tian and Burrage, 2005; Lay, 2000) The methods ensuing from this paradigm emphasize local interactions as opposed to a global description. The generated emergent behaviors are often surprisingly complex. Wolfram and others have shown such emergent features of cellular automata (Wolfram, 2002; Talia, 2000). In fire spread modeling, this was

* Corresponding author. Tel.: +33 0 473 40 50 19.
E-mail address: drch@isima.fr (D.R.C. Hill).

1364-8152/$ – see front matter © 2008 Published by Elsevier Ltd.
doi:10.1016/j.envsoft.2008.11.014

recently illustrated in (Porterie et al., 2007) comparing a cellular automata approach to experiments performed at the laboratory scale. At a field scale, in a strategy involving cellular automata, the coupling with atmospheric models and topographical models (stored in a Geographical Information System) shall be planned (Clark et al., 2004.) Furthermore, some new trends in fire safety research include modern data assimilation techniques (Cohen et al., 2007). All these new modular components of a simulator for wildfire at a field scale necessitate using techniques for high performance computing (the coarse-grained parallelization for data assimilation or atmospheric models). At the end, prediction tools of a wildfire behavior at fields scale require simulation times shorter than real times for an optimal management of firefighting.

Therefore, fire spreading modeling and simulation require efficient models and very high performance computations. The efficiency of discrete event simulators is beginning to receive significant attention (Wainer and Giambiasi, 2001; Lee and Kim, 2003; Hu and Zeigler, 2004; Muzy and Nutaro, 2005). Concerning cellular models, (Wainer and Giambiasi, 2001) show that the simulation of cellular models can be improved by "flattening" the hierarchy of coordinator objects which are used in the DEVS (Discrete EVent System Specification) (Zeigler, 1976; Zeigler et al., 2000a,b). In this flattened simulator, a single coordinator manages all the atomic components of a model. This can significantly reduce the cost of event routing, and it eliminates the need for multiple coordinator objects. Lee and Kim (2003) describe a similar solution that computes and stores possible event routes at compile time. Hu and Zeigler (2004) describe an improved scheduling algorithm for cellular models that are simulated using a hierarchy of coordinators and simulators. Muzy and Nutaro (2005) proposed a new simulation approach for DEVS models (Zeigler et al., 2000a,b) and Parallel Dynamic Structure Discrete Event (DSDEVS) models (Barros, 1997). The simulation architecture and communication protocol have been designed to improve efficiency by: (i) eliminating unnecessary simulator and coordinator objects, (ii) faster event scheduling by only storing references to active models, (iii) eliminating unnecessary internal synchronization messages, and (iv) eliminating unnecessary event routing messages.

From a modeling point of view, three kinds of approach have been developed so far through the DEVS formalism:

1. "Pure" DEVS models, in which cells are specified and simulated as usual atomic models (see Ntaimo et al., 2004) for a good example on fire spread),
2. A higher specification level has been developed through Cell-DEVS (Wainer, 2002). Timing mechanisms abstractions and geometrical sets have been added to usual DEVS structures,
3. Non-modular approaches to improve simulation efficiency (Muzy et al., 2003; Shiginah, 2006). In the latter very sound proofs of closure under coupling are provided. A simplified specification level is provided to facilitate modeling and improve simulation performances.

The third approach is definitely chosen from a modeling point of view (using a non-modular approach) and a simulation point of view, using simplified aggregated simulators: (Muzy et al., 2003). Furthermore, our fire spread model has already been simulated and modeled through the Cell-DEVS formalism (Muzy et al., 2002a,b, 2005; Wainer, 2006). The implementation was performed but we were not able to provide reasonable simulation times compared to the physical propagation time. Indeed, this fire spread model necessitates very small time and space discretizations leading to important computation overheads. Two reasons can explain this:

1. As B.P. Zeigler (DEVS' father) says: "(…) rule-based model specification (…) is achieved at some cost in execution time" (Zeigler, 1990). Moreover, dealing with discrete event timing mechanisms at a low level (here the cells), and for discrete-time simulation, leads to data structure overheads due to message exchanges. The next point justifies this overhead with comments from Shiginah, one of Bernie Zeigler's students:
2. "Cell-DEVS formalism, since it represents each cell as an atomic model, is considered as a conventional DEVS implementation of cell space models which has the performance drawback that is resulted by the huge volume of inter-cell communication generated during simulation. In addition, expressing cellular models in Cell-DEVS formalism is, to some extent, complex and requires more efforts at the modeler level. On the other hand, this dissertation introduces the multi-layer approach to simplify the modeling process and make the cell space's extensive specifications transparent to the end user." (Shiginah, 2006).

Hence, Active-DEVS aims to be an efficient model for simulating large-scale propagation phenomena. The software tools implemented have to keep pace with the rapid improvements of processing power of SMP machines. To reach these objectives, a cellular model (Worsch, 1997) and a simulation framework, founded on both DEVS (Zeigler et al., 2000a,b) formalism and object-oriented methodology, have been used.

An enhanced automaton models the propagation domain in which elementary behaviors describe each node. The spatial dynamics expression of the phenomena is thus facilitated. The DEVS formalism simplifies the modeling at a higher level and allows specifying components independently from automatically generated simulation algorithms. Hence, descriptions of the simulation treatments are facilitated.

The object-oriented architecture relies on design patterns and thus keeps a modular, elegant and adaptable design (Gamma et al., 1994.) A specific approach founded on an object container integrating parallel directives for SMP machines is performed. The implementation of this local parallelization allows improving execution times and is reported as fine-grained parallelization. The software simulation tool developed combines the experienced features of cellular automata, object-oriented methodology (design patterns), DEVS, and the efficiency of parallelization techniques based on OpenMP parallel compiler directives (OpenMP, 2002.) The presented approach is designed to evolve for predicting wildfires at field scale. This study therefore attempts evaluating the efficiency of the chosen fine-grained parallelization when combined with a modern object-oriented architecture, with cellular automata.

On the one hand, modeling is facilitated by the use of object-oriented techniques and DEVS. This technique and this framework have been used successfully in many different domains (for a review on DEVS applications, please refer to Zeigler et al., 2000a,b.) At the simulation level, design patterns are used to switch easily between different data structure implementations depending on cellular modeling requirements (more details are provided in sub-section 3.1) and parallelization techniques.

The main contributions of this approach can be summed-up as follows:

- A physics-based model is implemented in a consistent and adapted object-oriented and formal framework,
- The latest advances in the domain of discrete event modeling and simulation are used for optimization (at both modeling and implementation levels),
- SMP parallel implementations are included in a generic way,

Please cite this article in press as: Innocenti, E., et al., A software framework for fine grain parallelization of cellular models with OpenMP:..., Environ. Model. Softw. (2009), doi:10.1016/j.envsoft.2008.11.014

- The whole framework can be used in a very efficient and easy way by scientists for modeling and simulation on a single personal computer.

This paper is organized as follows: in section 2, the backgrounds for developing the simulation model are introduced; in section 3, the model and simulator designs are described; section 4 presents the parallelization technique conduced with the industry standard OpenMP, in order to enhance performance; in section 5, the details of the fire spreading implementation are explained and the simulation results are analyzed; finally, conclusion and perspectives are drawn in section 6.

2. Background

The cellular automata paradigm, the DEVS framework and design patterns are merged to achieve a multi-level specification of Active-DEVS.

2.1. Cellular automata

Cellular automata (CA) (Von Neuman, 1966; Wolfram, 2002) have been reinvented numerous times under different names and within diverse disciplines as counterpart to finite difference equations. Basically, a cellular automaton represents an array of identical cells, i.e., the cell space. The cell space is composed of individual cells. Each cell is a programmed automaton which interacts with others according to a fixed set of deterministic rules. Basic elements of a cellular automaton are: its state, the neighborhood of cells and its transition rule. The state is a variable associated to each cell in the cell space and represents the studied local data; the neighborhood represents the nearest cells used in the automata rules for calculating the behavior at the next time step; finally, the rules define how the state of a cell evolves with respect to the current state and the current states of its neighbors. At every time step states of cells are simultaneously updated according to the transition rule.

A basic cellular automaton is a structure:

$$\mathrm{CA} = \left(Z^d, S, N, \delta\right),$$

where, Z^d is the discrete lattice of d-tuples, S is the finite set of states, $N = \{(n_j = (x_{1j},\ldots,x_{dj}) | j \in \{1,\ldots,n\}\}$ is a finite ordered subset of Z^d (the neighborhood), δ: $S^{n+1} \rightarrow S$ is the transition rule of CA.

Although a variety of simulation models based on basic automata have been performed, simulating ecological processes require several evolutions of this fundamental computational model. The basic homogeneous cellular structure is too rigid to easily represent the diversity of processes interacting in an ecosystem. Generalizations of cellular automata methodology are then required (Bandini et al., 2001; Sipper, 1994). The combination of object-oriented programming and cellular automata paradigm offers the flexibility to include a large part of process description met in ecological modeling.

2.2. Design patterns

Design patterns are described in the literature as good solutions to common problems encountered when designing object-oriented architectures. They appear as powerful tools useful for environmental modeling. Nowadays, they are unavoidable when dealing with object-oriented programming. A lot of studies were founded on design patterns in the early 1990s, but the really famous ones are those developed by the "Gang of Four" (Gamma et al., 1994). These authors have identified and cataloged the design patterns from the object-oriented developments recognized by the software community in order to solve specific design problems. The key people who introduced design patterns in the software engineering community give this definition: patterns are "descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context." Appleton (2000) gives a clear picture of design patterns and provides pertinent information on their relevance to various programming problems they may wish to solve in ecological modeling. In the software design proposed in this paper, we aim at structuring reliable simulation systems by allowing optimized models to be composed in a flexible manner. A combination of design patterns, and a comparison between two enhanced cellular models for propagation phenomena were studied in a previous work (Innocenti et al., 2004a,b). In the following we only retain what we found to be best for fire propagation modeling.

2.3. Discrete EVent System Specification

Developed in the beginning of the 1970s, DEVS is an abstract universal formalism used for discrete event modeling (Zeigler, 1976).

A basic DEVS atomic model is a structure:

$$\mathrm{DEVS} = (X_M, Y_M, S, \delta_{\mathrm{int}}, \lambda, ta),$$

where, X_M is the set of ports and input values, Y_M is the set of ports and output values, S is the set of system's states, δ_{ext} is the external transition function, δ_{int} is the internal transition function, λ is the output function, ta is the time advance function.

Components of the model are presented via the descriptive variables. S represents the set of state variables. X is the set of input variables and Y is the set of output variables. An atomic model is influenced by internal and external events. Atomic model activity is described by the internal transition function δ_{int}, the output function λ and the external transition function δ_{ext}. External events are generated on the input ports of the atomic model and generate the model response to the outside. Internal events are programmed as a result of external events occurrences. The reader interested in more details will benefit from the following DEVS reference book (Zeigler et al., 2000a,b).

3. Model and simulator design

Separating model specifications from simulation algorithms improve productivity. Simulation algorithms are automatically generated. Models can be easily modified without modifying the corresponding simulators. The design of both models and simulators are presented hereafter.

3.1. Cellular object model

The combination of discrete event simulation techniques and the cellular automata paradigm enables us to reduce execution times restricting computations to active cells (i.e., cells changing state), a new kind of model, called Active-DEVS, is specified. The execution time gain can thus be considerable and the total execution time is less dependent on the size of the propagation domains. Indeed, only cells executing a state transition are considered for computation. Simulator activity is then limited to this set of active cells (or "active components".)

In fire spreading simulations, this technique permits the restriction of computations to fire fronts by indexing active components in memory (Innocenti et al., 2004a,b). The computing burden is thus limited to these components. To achieve this goal

Please cite this article in press as: Innocenti, E., et al., A software framework for fine grain parallelization of cellular models with OpenMP:..., Environ. Model. Softw. (2009), doi:10.1016/j.envsoft.2008.11.014

usual data structures are managed by the classical list algorithms implemented in numerous libraries. Such data structures are called *containers* thank to their ability to contain other objects. For the C++ language, retained for performance reasons (Stroustrup, 2000), the best known library for container management is the Standard Template Library, STL (Stephanov and Meng, 1995). A performance comparison between different containers used in fire spreading applications has been performed in (Innocenti et al., 2004a,b).

The objective is to develop a modular model structure adaptable to different execution contexts. Two design patterns are used in this scope: (1) the strategy pattern and (2) the adapter pattern. The Strategy Pattern is used to encapsulate the variants of algorithms which manage the active components and swap them strategically to optimize dynamically the management of active cells without changing the global framework architecture. The Adapter design pattern is used to adapt STL classes to our framework. Using this design technique, it is easy to switch the container and also to change between multiple algorithms which can be proposed in order to manage the active elements.

(1) Using the strategy design pattern, the container and the corresponding model management algorithm are separated and encapsulated into two separate classes : '*ActiveVector*' and '*Container*'. This enables an easy switching of the container algorithm according to simulation requirements. References to cells or events require managing data structures and event lists. The efficiency of a solution also depends on the application data. Some applications require managing time through schedulers (Hu and Zeigler, 2004), others do not and only require lists (Muzy et al., 2003). Concerning schedulers many solutions exist (unsorted arrays, binary heaps, trees, etc.) providing different algorithmic complexities according to the operations which deal with the elements of the scheduler (add, remove, get minimum time of the scheduler, etc.). For a more precise review, please refer to (Shiginah, 2006). According to the number of operations that will be required for a specific kind of model, there will be a best adequate solution. The cooperation between the strategy design pattern and the classes describing the active component transitions are illustrated in Fig. 1.

Whereas a classical derivation-based approach fixes the container algorithm, the strategy solution developed enables us to adjust it easily. The container algorithm can vary independently from the interface (class *ActiveVector*) used to manage the active components in the model. An abstract class is associated to the different categories of containers, and a concrete class to each specific container. The abstract class represents the common interface for the different kinds of containers; each concrete class uses this interface. The experimentation of different containers is facilitated, and the choice of a container depends on the particular simulation context. A *configuration* object indicates to the model what kind of container to choose. The active-components management thus defined is modular. The various STL containers have been tested in different configurations. Indeed, the efficiency of such object greatly influences the simulation performances.

(2) The *Adapter* design pattern is used to integrate the containers of the STL in a modular way (Stroustrup, 2000.) It is located between the *container* class, and the various possible implementations offered by the STL. Thus, the interfaces of the different containers available are converted into a generic interface (*ActiveVector* class). Fig. 2 describes both strategy and adapter design pattern cooperation. Intermediate classes (*listContainer*, *vectorContainer*, *dequeContainer*, *mapContainer*, *multiMapContainer*) delegate requests sent to them to the corresponding implementation (STL classes). The concrete container classes are derived from the virtual *container* class, each one containing a *container* type object. They are used to transmit the requests sent to the container object to the corresponding object of the STL.

With the design choices made, the next step is to select how to exploit this model on modern parallel computer architecture (bi and quad-cores) to enhance execution times. The local parallelization is performed into a Static Container (*StaticContainer* class),

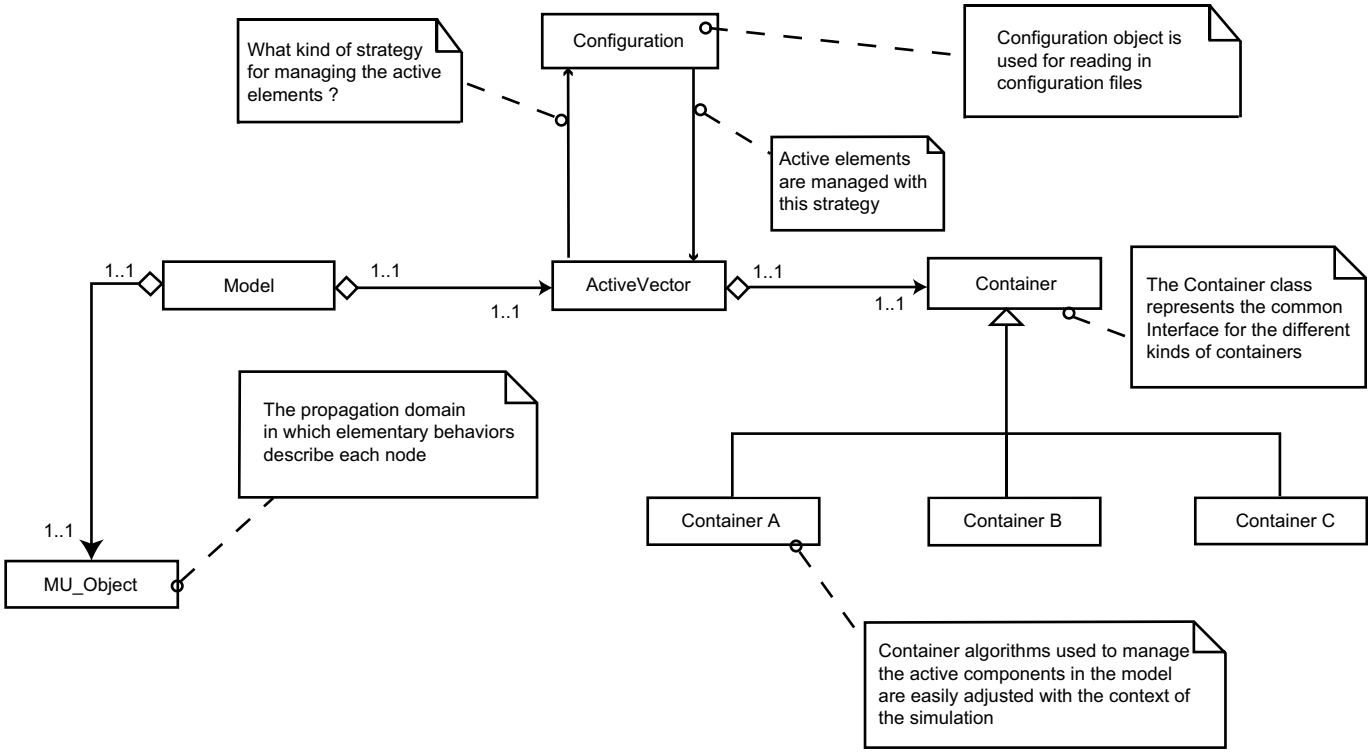

Fig. 1. Cooperation between the strategy design pattern and the *ActiveVector* object responsible for the transitions of the active components.

Please cite this article in press as: Innocenti, E., et al., A software framework for fine grain parallelization of cellular models with OpenMP:..., Environ. Model. Softw. (2009), doi:10.1016/j.envsoft.2008.11.014

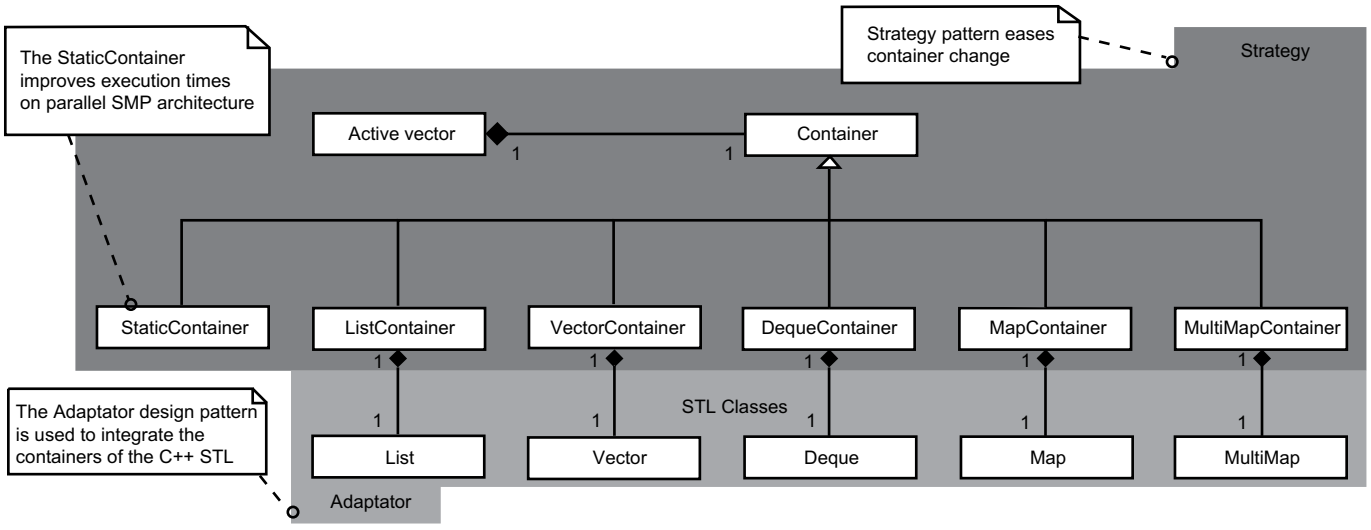

Fig. 2. UML collaboration diagram between the strategy design pattern and the adapter design pattern.

in which the OpenMP directives are implemented. Our own version of the container is developed in order to avoid any latency in element indexation which is dynamically performed in usual STL containers. Indeed, dynamic memory allocations are efficient but computationally expensive for fast simulations, alternative strategies are available. The Static Container is free from such dynamical memory management. A comparison between performances of static and dynamic containers is presented in (Innocenti et al., 2004a,b). The *StaticContainer* class is derived from the *container* abstract class.

3.2. Simulation framework

The simulation elements allow the model to evolve over time. The DEVS formalism facilitates the definition of the main objects of the simulation. The hierarchical and modular structure of a DEVS model is generally reflected in the specification of the model's simulator. The latter is automatically generated to activate the elements of the model (described in section 2.3). Each atomic model is associated with a simulator object. The simulator is controlled by sending messages such as "compute next state" and "compute next output", and it makes requests such as "get time of next event". A coordinator object is associated with each network model, and the coordinator can respond to the same types of messages as the simulator objects. The coordinator, as its name suggests, coordinates the execution of its component coordinators and simulators.

Coordinators and simulators are created differently according to a specified model. It is thus necessary to be able to build the same tree structure, for different implementations. The issue here is to make it possible for the root coordinator to choose a model, a coordinator and a simulator, then to build the object architecture of the simulation, using a simple interface. The solution relies heavily on the *abstract factory* design pattern, which is used to get round the limitations of inheritance as far as the programming complexity. The structure obtained facilitates the creation of the simulation models, as well as the simulators and coordinators related to them. The first prototype is written in C++, and is fully operational. Interfaces are developed using the *factory* design pattern. Instantiations of related models and processors are submitted to sub-classes. Three hierarchies of parallel classes are inter-connected: the *model* class, the *simulator* class and the *coordinator* class.

Fig. 3 represents the object interfaces developed with a modular design adaptable to different execution contexts.

This architecture is independent from the way of creating, composing and representing models, simulators and coordinators. The sequence diagram in Fig. 4 illustrates the instantiation procedure of the modeling components.

The root coordinator builds the simulation tree using as starting point the different families of cellular models and the available algorithms. A library of simulation tools is then made up. The parent root coordinator does not know the implementation classes of the families of models and processors (simulators and coordinators). The permutation between the latter becomes simple, and the consistency inside models and processors is reinforced. In that case, for addition or modification of the simulation encapsulated tools, only the *simFactory* objects are modified.

4. Implementation with OpenMP

The classes and relationships constituting the simulation framework have been presented. The design rules, which allow the development of modular model components, adaptable to different execution contexts, have also been detailed. This section describes how this framework is used for parallelizing the computations of active components using the OpenMP industry standard (OpenMP, 2002).

4.1. OpenMP

OpenMP is a portable standard for software parallelization on homogeneous architectures with shared memory (SMP) (Lucka and Sorevik, 2000). Thanks to a set of directives it is possible to give a simple description of the parallelization. There are two different kinds of parallel regions:

1 - Loops with independent iterations,
2 - Independent code blocks.

In this experiment, only a loop parallelization has been used. The goal of the OpenMP standard is to increase portability of parallel programs intended for shared memory architectures. Specifications of OpenMP are decided by the "OpenMP Architecture Review Board." In OpenMP, the executions of parallel loops are based on the fork-join programming model. In a parallel section the thread running is divided into groups of threads, which are

Please cite this article in press as: Innocenti, E., et al., A software framework for fine grain parallelization of cellular models with OpenMP:..., Environ. Model. Softw. (2009), doi:10.1016/j.envsoft.2008.11.014

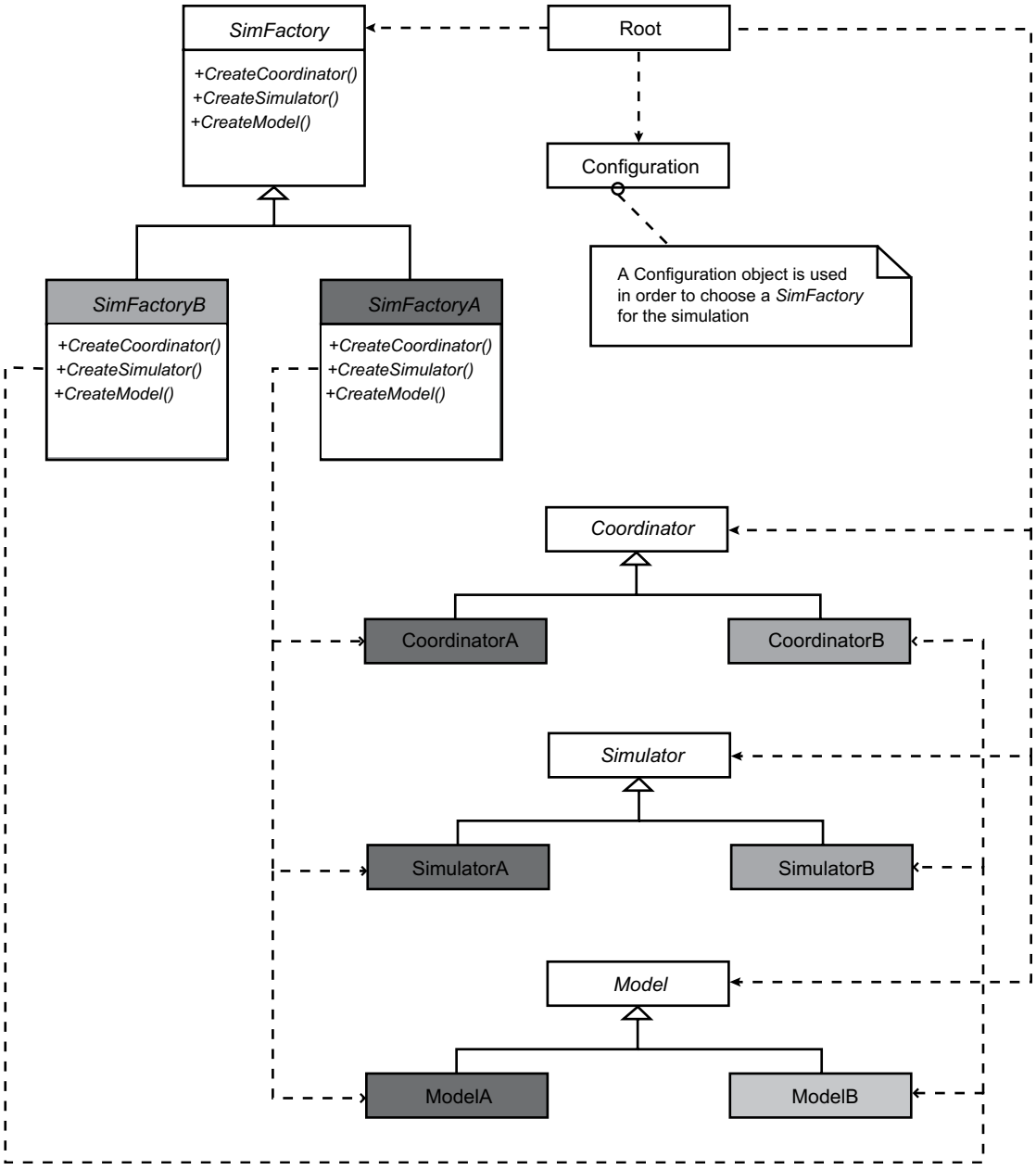

Fig. 3. The abstract factory design pattern in the simulation framework.

synchronized at the end of the section and finally joined into only one thread (the original one). OpenMP makes it possible to specify easily sharing and synchronization of tasks, providing directives which have the same syntax. It should be noticed that currently, OpenMP does not propose directives which distribute data (clustering). However, with the increasing number of SMP architectures in high performance clusters (HPC), the current tendency consists in developing hybrid solutions mixing OpenMP and MPI. Thus, cluster nodes correspond to SMP machines whose iteration loops use OpenMP, and communications between nodes are performed through MPI (He and Ding, 2002; Henty, 2000).

4.2. Fine grain parallelization of active-components

The object-oriented architecture developed allows focusing on the computations of active components. This management is centralized in containers. The latter are subdivided and their treatments parallelized to improve the execution times (local parallelization). However, the parallelization of the treatments associated to the management of the active components requires the use of a shared memory. Indeed, the strong coupling existing between the list of active components and the cellular domain components requires a common memory space to facilitate information exchanges during computation. It is then mandatory to choose parallelization methods based on the use of a shared memory. Explicit environments of parallelization such as MPI (MPI, 1995), or PVM (Geist et al., 1994) are not adapted for this kind of parallelization. In this case the model structure requires using specific mechanisms for message exchanges. These mechanisms slow down the simulation execution. Thus, it is preferable to parallelize the container using lightweight processes which share a common memory such as POSIX threads (ANSI, 1994), or implicit parallelization environments founded on POSIX threads, as OpenMP (OpenMP, 2002).

Please cite this article in press as: Innocenti, E., et al., A software framework for fine grain parallelization of cellular models with OpenMP:..., Environ. Model. Softw. (2009), doi:10.1016/j.envsoft.2008.11.014

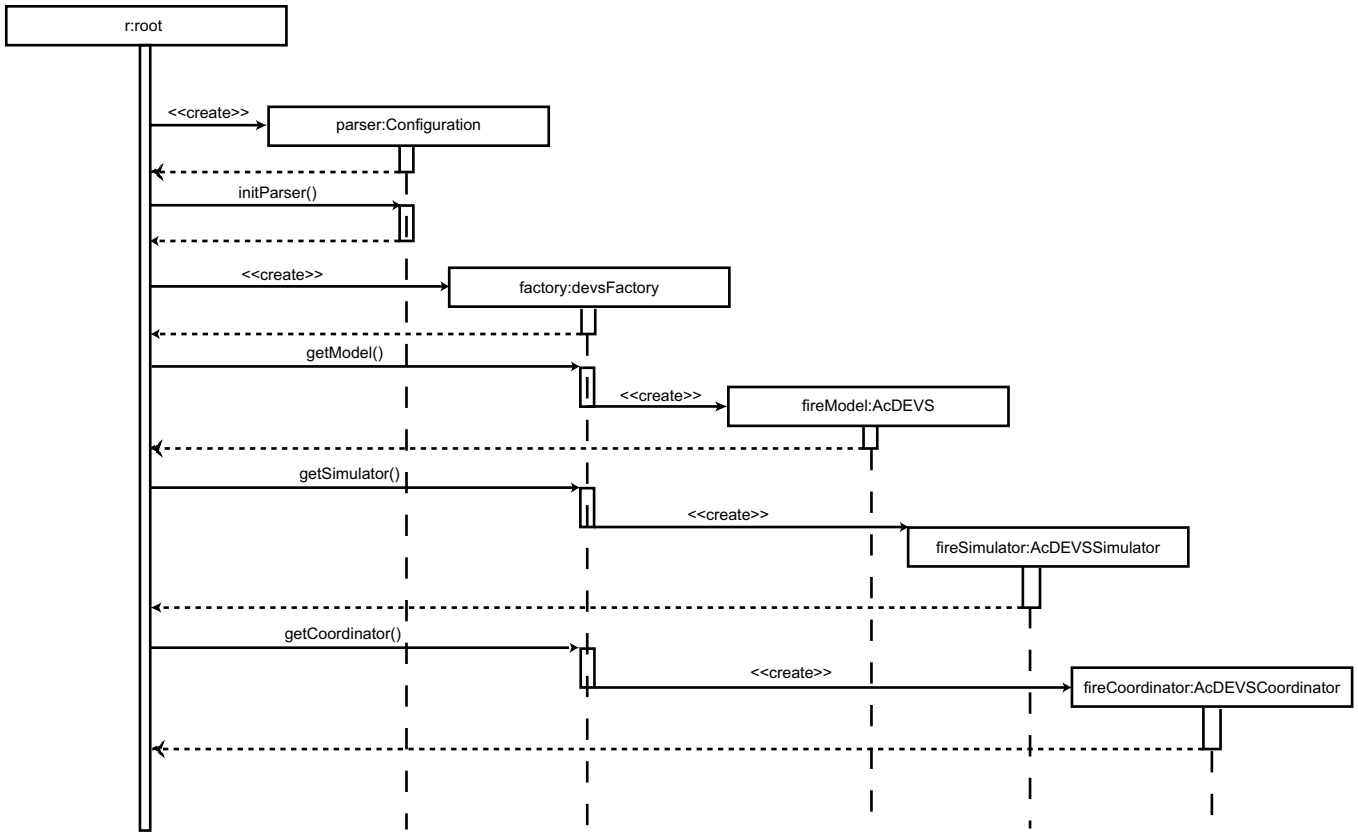

Fig. 4. Coordination of modeling and simulation components with the abstract factory: instantiation procedure of the modeling components.

To improve the management of active components the container is divided into components sub-sets. Then, every sub-set is encapsulated in a lightweight process. Processes reference active components. Every process is responsible for component state transitions and the update of the domain part it represents. The container division is illustrated in Fig. 5.

All the cellular components are updated in a synchronous way, hence, it is crucial to define synchronization barriers to guarantee the simulation consistency.

4.3. Application to fire spreading

Developing an efficient fire spread model which reproduces the details remains a challenge for research. Models from physics are currently used to improve the pertinence of fire spread models and associated decision aid. The physical model in use (Eq. (1)) has been validated in several laboratory experiments (Santoni et al., 1998; Dupuys, 1995). The model is based on experimental data (evolution of a flame front in a domain of 1 m^2 of pine needles, without slope, nor wind). However, the model will be tested in much larger scale areas. Hence, whatever the scale, the fire phenomenon is considered as a transport phenomenon expressed in a reaction–diffusion equation. The current model can be extended to a higher scale implying many other control parameters such as vegetation structure and composition, ground topography, local wind, etc. The laboratory fire first considered here enables us to evaluate without difficulty the combination of: object-oriented techniques and fine grain parallelization (from a computational point of view), as well as physics-based modeling of fundamental propagation mechanisms of fire spreading.

Extending the present laboratory fire simulation to field scale requires the development of a more precise model incorporating thermal, vegetal mass and wind velocity additional components. Implementing more control parameters into the model to match the experiment will necessitate prior validation of the simulation tool developed. According to this scope, this experiment has been carried out using laboratory scale fire data, allowing the calibration and validation of the proposed framework. The flexibility and the modularity of the architecture is a preliminary step before implementations of more precise model components. However, the simulation architecture is validated here. Field scale modeling and simulation will only necessitate adding supplementary details preserving the whole structure of the simulator.

In this scope, the physical mechanisms describing the propagation, identified and modeled on laboratory scale, are supposed to remain valid on large-scale areas. A preliminary numerical study makes it possible to provide an approximated numerical solution of the model, using both finite differences for spatial derivatives and an explicit scheme for the time advancement of the solution. The propagation domain is then subdivided into elementary components which constitute the ground and the plants, each one being described by the following algebraic:

$$T_{i,j}^{k+1} = aT_{i-1,j}^{k} + aT_{i+1,j}^{k} + bT_{i,j+1}^{k} + bT_{i,j+1}^{k} + cQ\left(\frac{\partial\sigma}{\partial t}\right)_{i,j}^{k-1} + \mathrm{d}T_{i,j}^{k} \tag{1}$$

where, $T_{i,j}$ represents the temperature of a node of the grid. Coefficients a, b, c and d depend on the time step and the space step considered (Santoni, 1996). Parameters of the model are given starting from experimental statements of temperatures obtained according to time. The cellular division of space generates problems in extreme cases. In order to solve them a fixed value is given on the edge cells (Coquillard and Hill, 1997.) Numerical results were

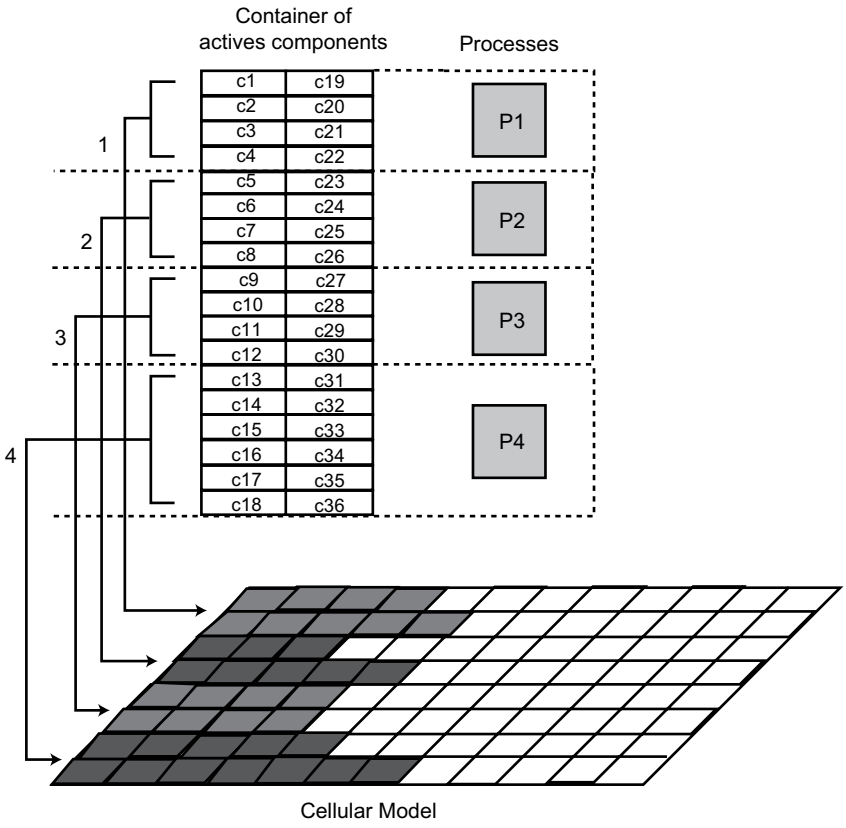

Fig. 5. Division of a container of active components.

compared with the experimental data for various ignitions and the quality of predictions is remarkable (Santoni et al., 1998).

However, the precision of these models make them difficult to be simulated under realtime constraints. Moreover, fire spread complexity requires refining progressively model specifications according to the simulation results. The corresponding simulation code has to be modified easily to reduce both implementation and testing phases. The object-oriented framework developed allows:

(1) To easily integrate model modifications as research advances (new ground vegetation, influence of both slope and wind, etc.),
(2) To obtain efficient execution times with a precise model running faster than a real experiment,
(3) To optimally exploit the inherent parallelism of cellular models (Jorba et al., 2002).

The simulation components designed support the evolutionary nature of the structure. An example of fire spread simulation using an Active-DEVS model is illustrated in Fig. 6.

4.4. OpenMP implementation

The OpenMP parallelization standard gives a set of directives, which allow describing simply the local parallelization of the container of active components (as illustrated in the extract of C++ code of Fig. 7). These directives are comments activated by the compiler thanks to the use of an adequate option. A compiler which does not support the OpenMP standard will not pay attention to them. Hence, the code is portable and the maintenance is limited to a unique version.

The *schedule*() clause defines the scheduling mode of processes concerned with the *For* directive. The "runtime mode" is used to differ the scheduling at runtime according to the value of the environment variable OMP_SCHEDULE. No default scheduling mode is proposed by OpenMP. The choice of the runtime mode allows to balance the computing as referred by the OMP_SCHEDULE value chosen by the computing center. The interested reader should refer to the OpenMP courses (Gondet and Lavallée, 2000.)

At the entry of a parallel area delimited with OpenMP directives, the master process generates lightweight processes which execute

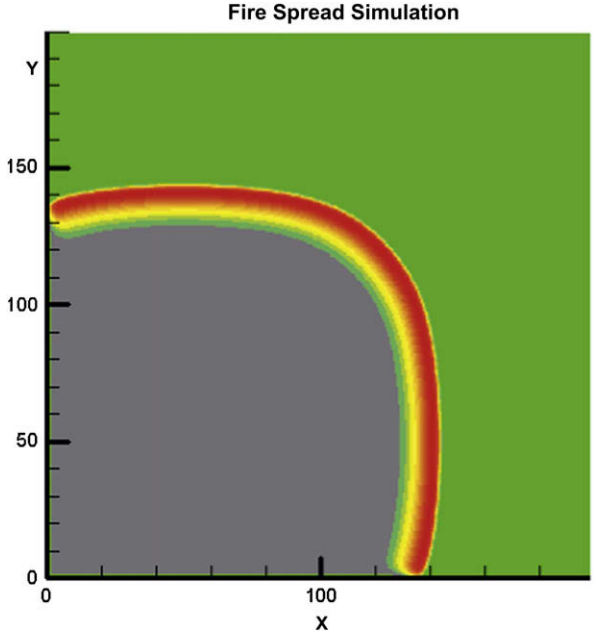

Fig. 6. Overview of a fire spread simulation.

```
...
//For Loop
#pragma omp parallel
{
   #Pragma omp for schedule(runtime)
    //Transitions of active components
    for(cursor=deb; cursor<=fin; cursor++){
        activeCell=SetOfActiveCells->getActiveElt(cursor);
       if(activeCells){
           updateCellNextTemperature(activeCell);
       }
   }
}
//Delete inactives cells
SetOfActiveCells -> removeActiveElt();
...
```

Fig. 7. Excerpt of C++ code implementing OpenMP directives.

a task in parallel. In this experience, parallelization consists in executing the main loop of the container, distributing the iterations between the different lightweight processes (loop level parallelism). This parallel model has been simulated on a node of the Zahir parallel architecture of the "Institut du Développement et des Ressources en Informatique Scientifique (IDRIS)." This node has *32* processors *Power4 P690* running at *1.3* Ghz, equipped with *256* Gb of memory for a peak power announced at *166.4* Gflops/s. This machine is exploited with AIX – version *5.1*. Jobs are submitted in batch mode to obtain exclusively the number of processors required.

5. Simulation results

The Simulation results presented in this section concern experimental fires conducted on Pinus Pinaster litter, in a closed room without any air motion, at the INRA (Institut National de la Recherche Agronomique) laboratory near Avignon, France. The initial experiments were performed to observe fire spread for point-ignition and line-ignition fires under no slope and no wind conditions (Balbi et al., 1998). The experimental apparatus was composed of a one square meter aluminum plate protected by sand. A porous fuel bed was used, made up of pure oven dried pine needles spread as evenly as possible on the total area. These experiments are simulated here under various conditions. Various simulation models have been designed first using sequential programming (optimizing the simulator structure, the container implementations and accounting only for active burning cells during the simulation). Once the sequential simulation has been optimized, parallel programming has been considered.

A first simulation is designed through the Cell-DEVS formalism (Wainer and Giambiasi, 2001) and implemented in the CD++ environment (Wainer, 2002). The laboratory experiment consists of a combustion table of 30 cm long and 60 cm wide for a line-ignition, the prediction of spread rate (2.96 mm/s) and the propagation are in agreement with the experimental data for every approach (cf. Fig. 8). The black lines represent the position of the experimental isothermal line of 300 Celsius (ignition interface).

In Fig. 9, the execution times of the Cell-DEVS simulation are plotted for different square cell domains and for a real propagation of 3 s. CD++ execution times are definitely better than in a previous JDEVS one (where the simulation of a 100 × 100 cells domain was not possible). In both simulations, the discrete event cells exchanging events embedded in messages compute their temperature evolution at every time steps. Only cells changing state send events activating their neighbors. Even if this modeling and simulation experiment is elegant and grounded formally, the numerous messages exchanged produce overhead because of data structure

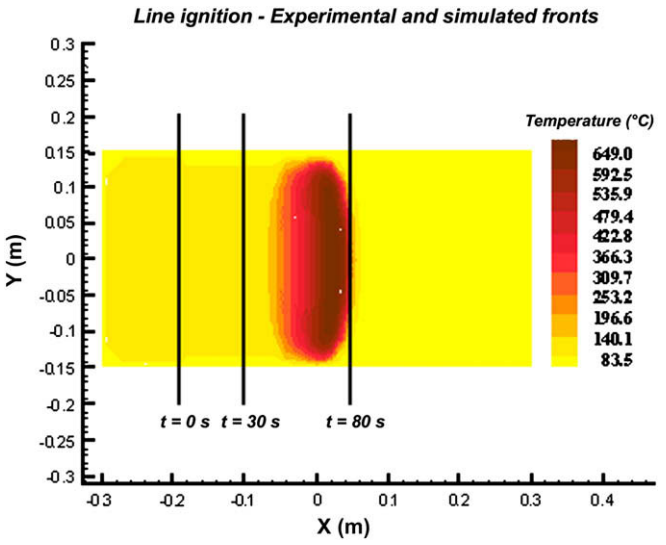

Fig. 8. Simulated and experimented temperatures field representation of a line-ignition.

management (of schedulers). This results in an execution time much greater than the actual propagation time. Hence, a simulator structure closer to the model structure and focusing directly on active cells, has been designed (Muzy et al., 2003).

As depicted in Fig. 10, a point-ignition has been simulated by initializing center cells with a temperature gradient. Fire spreads circularly and symmetrically up to the plate borders. Fig. 11 depicts the simulation time gain for different temperature gradients and a real propagation of *200 s*. With a basic simulation (which does not focus on active elements), the 200 s of real propagation are simulated in *160 s*. A temperature gradient is progressively adjusted for the detection of activity of cells. When the temperature of a cell begins to increase, the temperature of the cell is compared to the gradient. If the temperature is greater than the gradient, the cell is considered to be active and added to the container tracking active cells. If the temperature is less than the gradient, the cell is considered to be inactive and ignored from the main simulation loop. For a temperature gradient of only *1 K*, the performance improves by *20 s*. From a temperature gradient of *10–30 K*, the performance gain remains about *30 s*. The simulation time drops to *100 s* for temperature gradients greater than *40 K*.

A new experiment has been designed to compare static simulations (which do not account for active components) and

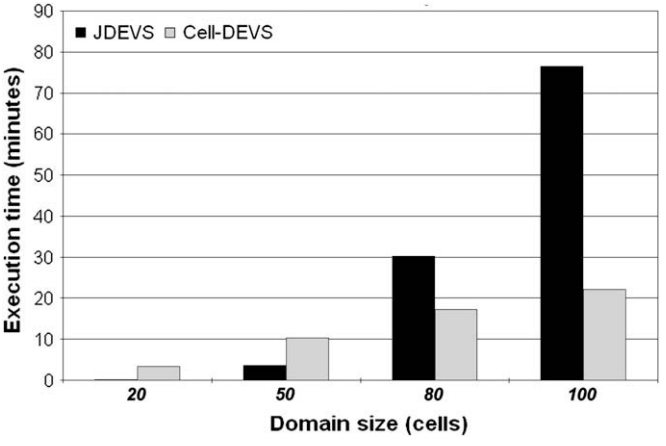

Fig. 9. Initial and hierarchical comparison.

Please cite this article in press as: Innocenti, E., et al., A software framework for fine grain parallelization of cellular models with OpenMP:..., Environ. Model. Softw. (2009), doi:10.1016/j.envsoft.2008.11.014

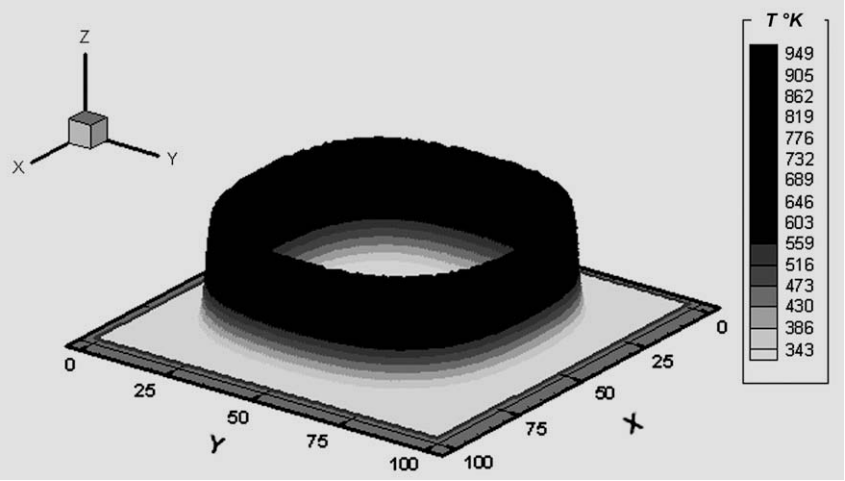

Fig. 10. Fire spread simulation of a point-ignition.

simulations tracking active components through various STL implementations. Punctual and linear ignitions with different temperature gradients on heterogeneous domain of 40,000 components have been simulated. In static simulation, the container saves the references of the fire front components in a static array. The static container is declared before execution in memory and its size remains fixed during the simulation. In the case of the activity tracking approach the management of the fire front components is done using STL containers. STL data structures organize and manage dynamically the active components, i.e. the size of the container change during the simulation. The static container and STL containers used are compared in various experiments which are depicted in Table 1. The containers of the experiment can be divided into two categories, sequence containers (list, vector and deque), and associative containers (map and multimap). The list container allows fast insertions and deletions anywhere in the container, but it is not possible to randomly access an element. The vector cainer allows managing a sequence of elements as a contiguous block of storage. Each block is implemented as an array that grows on demand. The deque (double-ended queue) container allows fast insertions and deletions at the beginning and at the end of the container and allows to randomly access any element quickly. The map container consists of a key/value pair. The key is used to order the sequence, and the value is associated with that key. The map only allows one instance of a key or element to be inserted into the container. The multimap container allows multiple instances of elements. For more information on the containers the reader will benefit from the following reference (Nicolai, 1999).

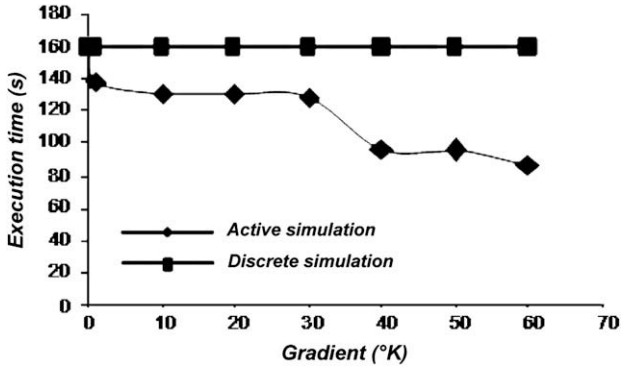

Fig. 11. Execution time gain over temperature gradient of the test for a real propagation of *200 s*.

The results in Fig. 12 show that the critical aspect of the algorithm is mostly related to the management of the front fire components. The simulation process spends a greater part of its computation time in the fire front management, and the container effectiveness improves simulation times. The List Container is not exploitable because the simulation times obtained are up to a day. Map container and multimap container are not effective; their simulation times are twice the static container, vector container and deque container simulation times. The comparison between static approach and dynamic approach is made on the basis of the static container and vector container, because deque container is not adapted here. If the fire front is limited, i.e. if the number of active component is limited, the static approach is faster than the dynamic approach (experiment 1), in all other cases, the vector container is slightly more effective (experiments 2–4). Static container and vector container lend themselves particularly good data structure to manage active elements of the fire front.

A final experiment has been designed to test the efficiency of the parallelization of the last efficient sequential implementation. The experiment presented in this section simulates a line ignition through a homogeneous domain constituted by *50,000* (250×200) components corresponding to a pine needle experiment.

Simulation results are reported onto the graph of Fig. 13, function of the number of processes, for a virtual time $t_{\text{simulation}} = 600$ s. Parallel simulations are carried out with *2*, *4*, and *8* processors, and executed five times to obtain average execution times.

The next Fig. 14, presents the speedup $S(P)$ obtained according to the number of processes P, such as:

$$S(P) = \frac{T(1)}{T(P)},$$

where, $T(P)$ represents the execution time obtained with P processes.

Table 1
Description of the experiments.

Heterogeneous domain of 40,000 components	Ignition type	Number of front fire
Experiment 1	Point ignition	1
Experiment 2	Line ignition	1
Experiment 3	Point ignition	2
Experiment 4	Line ignition	2

Please cite this article in press as: Innocenti, E., et al., A software framework for fine grain parallelization of cellular models with OpenMP:..., Environ. Model. Softw. (2009), doi:10.1016/j.envsoft.2008.11.014

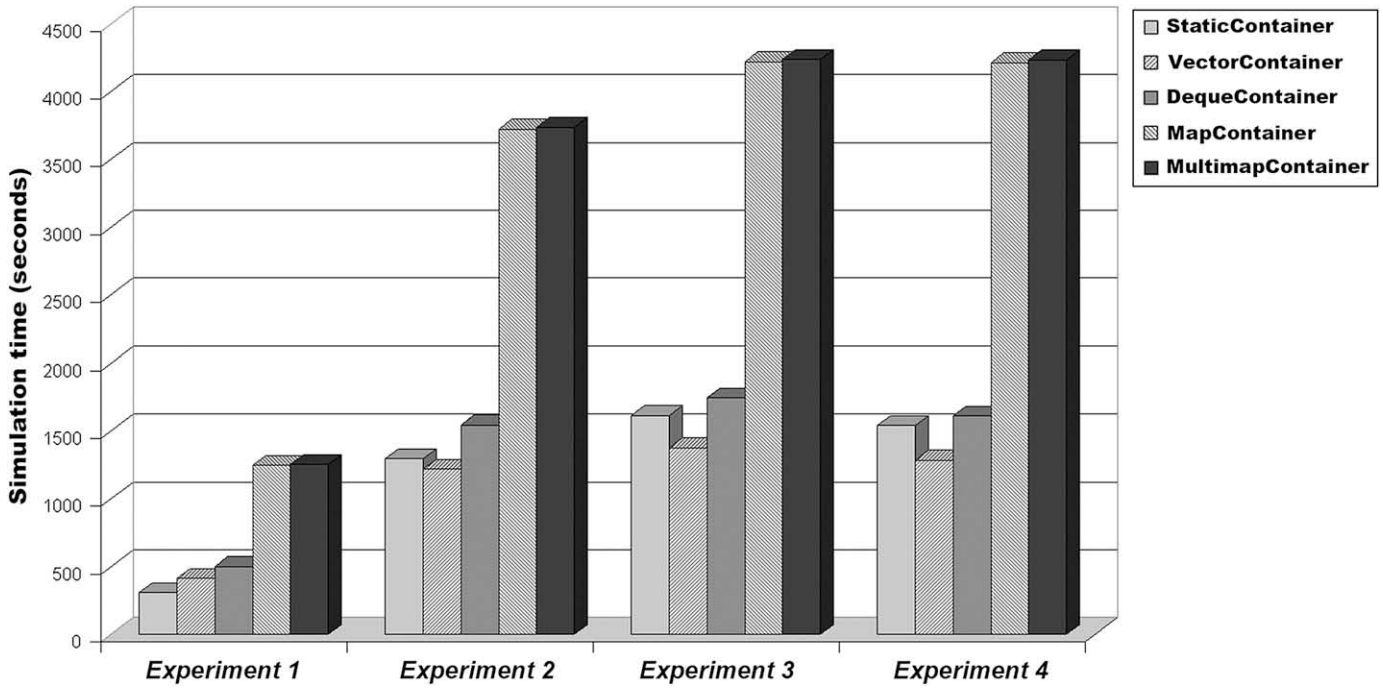

Fig. 12. Comparison of containers effectiveness.

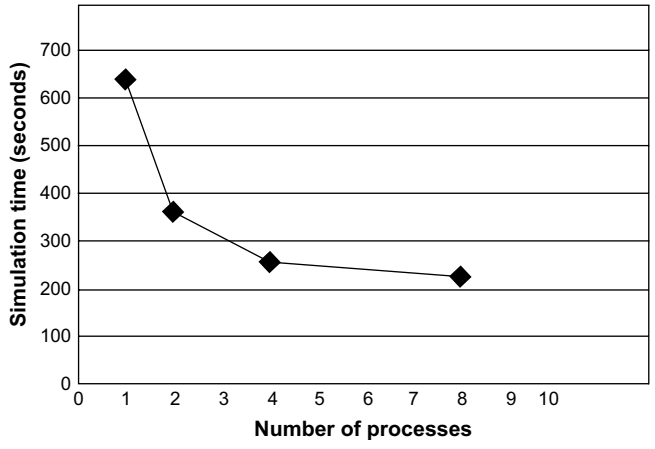

Fig. 13. Execution times obtained for different numbers of OpenMP lightweight processes.

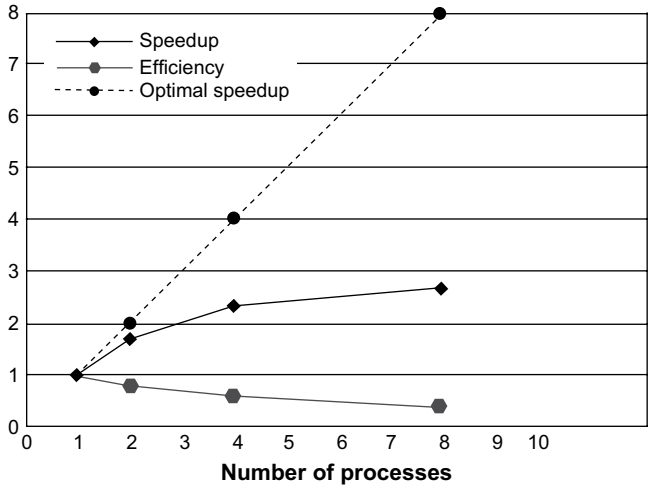

Fig. 14. Execution curves of speedup and effectiveness obtained with OpenMP.

The effectiveness of the simulation is also calculated. It is given by:

$$E(P) = \frac{S(P)}{P}.$$

Speedup and effectiveness curves corroborate the assumption according to which the parallelization of the container of active components allows to reduce execution times. This optimization is directly efficient with bi-processors and quadri-processors, and this is particularly interesting for recent cellular processors with duo or quad-cores. With more than 8 processors, this approach quickly reaches an asymptote. This predictable result according to our design is dictated by the Amdhal Law (Jordan and Alaghband, 2002). Indeed, the addition of processors does not always produce a linear factor of improvement for performance, particularly when models require local communications.

As a conclusion, the OpenMP compilation directives allow to reduce significantly execution times with a fine grain parallelization on a small number of processors. Furthermore, these results tend to confirm that parallelization techniques must be used to provide a decision tool able to help fire fighters during a forest fire. In that case, a decision tool has to provide fire front positions faster than the fire propagates in reality. The execution of this configuration on only one processor requires an average time of $t_{\text{execution}} = 658\ s$, which is, higher than the virtual time $t_{\text{virtual}} = 600\ s$ which represent the propagation time for a real fire. In this example, the use of OpenMP directives allows us to decrease execution times under this virtual time limit, in a simple and fast way.

6. Conclusion and perspectives

The theory of modeling and simulation and object-oriented programming provides methods and techniques worth applying when modeling complex spatial phenomena. More precisely, cellular automata, design pattern and DEVS are used in this study.

Please cite this article in press as: Innocenti, E., et al., A software framework for fine grain parallelization of cellular models with OpenMP:..., Environ. Model. Softw. (2009), doi:10.1016/j.envsoft.2008.11.014

Based on them, a new kind of model, called Active-DEVS, has been specified and used in fire spreading simulations.

Modeling spatially complex systems from the fundamental computational model of cellular automata (CA) improve the simulation treatments. In fact, CA make it possible to express the space dynamics of the phenomena and to apprehend the complexity of some real systems. The concurrent execution of the treatments of the simulation is facilitated with the association of the techniques of parallel discrete events simulation. The DEVS formalism helps for developing the Active-DEVS computing model. The latter can be considered as an optimization which puts the focus on the active cells of a CA. It is used in a simulation object-based software framework in which the rules and the architecture improve the process of development in terms of time, software legibility, evolution, and portability. Besides, designing a reusable, extensible, and adaptable framework is a difficult task and design patterns were used to help achieving this goal.

OpenMP directives and SMP architectures provided a suitable, evolutionary and powerful support. The object-oriented framework has been designed to provide the basic components for the construction of discrete event cellular simulations, to be flexible and extensible, and to integrate the Active-DEVS model.

The interest of OpenMP directives to speedup the simulation for bi-processors and quadri-processors has been pinpointed here. It enables the decreasing of execution times under the real time deadlines of fire spreading. This speedup, based on a fine grain parallelization, also shows its limits. It is not designed for a large number of processors. Our approach is to consider this optimization as part of a parallelizing methodology in which fine grain parellizations can be applied on local nodes of computing clusters or computing grids. A coarse grain parallelism can then divide space and maps to handle large-scale landscape and assign parts of a landscape to a large number of nodes, each node being capable of local parallelism (duo or quad-core.) In this classical "divide and conquer" strategy, this study has shown that local simulations are able to benefit from our fine grain optimization.

We also plan to improve the capabilities of the framework and to model new physical experiments. With the acceptance of message passing standards such as MPI (MPI, 1995) it is well suited to join both OpenMP and MPI, i.e., to combine shared memory programming on shared memory nodes with message passing communications between these nodes (Sarkar et al., 2006; Henty, 2000; Badern and Jaja, 1999). In fact, these hybrid architectures encourage a hybrid parallel programming paradigm. They rely on Distributed Shared Memory (DSM) systems which implement shared memory abstraction on a network of workstations (Lu et al., 1998; Hess et al., 2002). This will be achieved using the Active-DEVS model and supporting framework as the foundation for a larger parallel and distributed simulation environment, which is currently under development.

Acknowledgments

The authors would like to acknowledge the technical assistance of the French National computing center (IDRIS), for their availability and their help. This project is supported by a research grant from the Regional collectivity of Corsica.

References

ANSI, 1994. American National Standards Institute. IEEE standard for information technology: Portable Operating System Interface (POSIX). Part 1, system application program interface (API) – amendment 1 – realtime extension [C language].

Appleton, B., 2000. Patterns and Software: Essential Concepts and Terminology Available from: http://www.cmcrossroads.com/ bradapp/docs/patterns-intro.html.

Badern, D.A., Jaja, J., 1999. Simple: a methodology for programming high performance algorithms on clusters of symmetric multiprocessors (SMPs). Journal of Parallel and Distributed Computing 58, 92–108.

Balbi, J.H., Santoni, P.A., Dupuy, J.L., 1998. Dynamic modelling of fire spread across a fuel bed. Int. J. Wildland Fire 1998, 275–284.

Bandini, S., Mauri, G., Serra, R., 2001. Cellular automata: from a theoretical computational model to its application to complex systems. Parallel Computing 27, 539–553.

Barros, F.J., 1997. Modeling formalisms for dynamic structure systems. ACM Transactions on Modeling and Computer Simulation 7 (4), 501–515.

Clark, T.L., Coen, J.L., Latham, D., 2004. Description of a coupled atmosphere-fire model. International Journal of Wildland Fire 13, 49–63.

Cohen, J.L., Beezley, J.D., Bennethum, L.S., Douglas, C.C., Kim, M., Kremens, R., Mandel, J., Quin, G. Vodacek, A., 2007. A Wildland Fire Dynamic Data Driven Assimilation System. In: 11th Symposium on Integrated Observing and Assimilation Systems for the Atmosphere, Oceans, and Land Surface (IOAS-AOLS).

Coquillard, P., Hill, D.R.C., 1997. Modélisation et Simulation d'Ecosystèmes. Des modèles déterministes aux simulations à évènements discrets Masson.

Dupuys, J.L., 1995. Slope and fuel load effects on fire behavior: laboratory experiments in pine needles fuel beds. International Journal of Wildland Fire 53, 153–164.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1994. Design patterns. Elements of Reusable Object-Oriented Software. Addison Wesley, ISBN 0-201-63361-2.

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. In: Kowalik, Janusz (Ed.), Scientific and Engineering Computation. MIT Press Available from: http://www.netlib.org/pvm3/book/pvm-book.html.

Gondet, E., Lavallée, P.F., 2000. OpenMP : concepts et présentations. Available from: http://www.idris.fr/.

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

Henty, D.S., 2000. Performance of Hybrid Message-Passing and Shared-Memory Parallelism for Discrete Element Modeling. In: Proceedings Supercomputing 2000, November 4–10. IEEE Computer Society, Dallas, Texas, USA, ISBN 0-7803-9802-5, pp. 1–9. CD-ROM.

Hess, M., Jost, G., Matthias, S.M., Rühle, R., 2002. Experiences using OpenMP based on Compiler Directed Software DSM on a PC Cluster, WOMPAT2002: Workshop on OpenMP Applications and Tools. Arctic Region Supercomputing Center, University of Alaska, Fairbanks, August 5th–7th, pp. 211–226.

Hu, X.B.P. Zeigler, 2004. A high performance simulation engine for large-scale cellular DEVS models. In: High Performance Computing Symposium (HPC'04), Advanced Simulation Technologies Conference (ASTC), Arlington, USA, pp. 3–8.

Innocenti, E., Muzy, A., Hill, D.R.C., Aïello, A., Santucci, J.F., 2004a. Active-DEVS: a computational model for the simulation of fire propagation. In: Proceedings IEEE, SMC 2004, International Conference on Systems, Man and Cybernetics, October 10–13. The Hague, The Netherlands, pp. 1857–1863.

Innocenti, E., Muzy, A., Aïello, A., Santucci, J.F., 2004b. Discrete Event Simulation of fire spread: a modular Approach for the Choice of a Strategy of Active Elements Management. In: Dans Proceedings International Carpathian Control Conference ICCC'2004, Zakopane Poland, pp. 69–74.

Jorba, J., Margalef, T., Luque, E., Campos da Silva André, J., Viegas, D.X., 2002. Parallel Approach to the Simulation of Forest Fire Propagation. In: Proceedings Environmental Communication in the Information Society. 16th International Conference "Informatics for Environmental Protection". Vienna University of Technology, pp. 69–81. September 25–27.

Jordan, H., Alaghband, G., 2002. Fundamentals of Parallel Processing. Pearson Education. Prentice Hall.

Karafyllidis, I., Thanailakis, A., 1997. A model for predicting forest fire spreading using cellular automata. Ecological Modeling 99, 87–97.

Langlois, A., Phipps, L., 1997. Automates cellulaires, application à la simulation urbaine. Edition Hermes.

Lay, J.G., 2000. A Land Use Change Study using Cellular Automata. In: Proceedings of 21st Asian Conference on Remote Sensing, Taipei, Taiwan. Available from:http://www.gisdevelopment.net/aars/acrs/2000/ts12/ts12003.shtml).

Lee, W.B., Kim, T.G., 2003. Simulation Speedup for DEVS Models by Composition-Based Compilation. Summer Computer Simulation Conference, Montréal, Canada, pp. 395–400.

Lucka, M., Sorevik, T., 2000. Parallel Wavelet-Based Compression of Two-dimensional Data. In: Proceedings of Algorithmy 2000. Conference on Scientific Computing, pp. 227–235.

Lu, H., Hu, Y.C., Zwaenepoel, W., 1998. OpenMP on networks of workstations. In: Supercomputing '98, pp. 1–15.

Mell, W.E., Jenkins, M.A., Gould, J., Cheney, P., 2007. A Physics-based approach to modeling grassland fires. International Journal of Wildland Fire 16 (1), 1–22.

Forum, 1995. MPI: a Message-Passing Interface Standard. Technical report, University of TennesseeMessage Passing Interface, Knoxville, TN. Version 1.1

Please cite this article in press as: Innocenti, E., et al., A software framework for fine grain parallelization of cellular models with OpenMP:..., Environ. Model. Softw. (2009), doi:10.1016/j.envsoft.2008.11.014

Morvan, D., Dupuy, J.L., 2004. Modeling the propagation of a wildfire through a Mediterranean shrub using a multiphase formulation, 2004. Combustion and Flame 138 (3), 199–210.
Muzy, A., Nutaro, J.J., 2005. Algorithms for efficient implementation of the DEVS & DSDEVS abstract simulators. In: First Open International Conference on Modeling and Simulation (OICMS), Clermont-Ferrand, France.
Muzy, A., Innocenti, E., Wainer, G., Aiello, A., Santucci, J.F., 2002a. Cell-DEVS Quantization Techniques in a Fire Spreading Application, Winter Simulation Conference (WSC) – Exploring New Frontiers. IEEE/ACM/SIGSIM/SCS, San Diego, USA, pp. 542–549.
Muzy, A., Wainer, G., Innocenti, E., Aiello, A., Santucci, J.F., 2002b. Comparing simulation methods for fire spreading across a fuel bed, Simulation and planning in high autonomy systems conference (AIS), SCS, Lisbon, Portugal, pp. 219–224.
Muzy, A., Innocenti, E., Hill, D., Santucci, J.F., 2003. Optimization of cell spaces simulation for the modelling of fire spreading, 36th Annual Simulation Symposium (ANSS). IEEE/SCS/ACM, Orlando, USA. 289–296.
Muzy, A., Innocenti, E., Wainer, G., Aiello, A., Santucci, J.F., 2005. Specification of discrete event models for fire spreading, Simulation: transactions of the society of modeling and simulation international. SCS 81, 103–117.
Ntaimo, Zeigler, B.P., et al., 2004. Forest fire spread and suppression in DEVS. Simulation 80, 479–500.
Nicolai, M., 1999. The C++ Standard Library: a Tutorial and Reference. Addison-Wesley Professional.
OpenMP, 2002. Official OpenMP Specifications. C/C++ Version 2.0 Available from: http://www.openmp.org/.
Porterie, P., Zekri, N., Clerc, J.P., Loraud, J.P., 2007. Modeling forest fire spread and spotting process with small world networks. Combustion and Flame 149 (1–2), 63–78.
Santoni, P.A., Balbi, J.H., Dupuy, J.P., 1998. Dynamic modeling of upslope fire growth. International Journal of Wildland Fire 9 (4), 285–292.
Santoni, P.A., 1996. Propagation des feux de forêts : Modélisation dynamique et résolution numérique, validation sur des feux de litière. PhD thesis, Università di Corsica – Pasquale Paoli.
Sarkar, A., Benabbou, N., Ghanem, R., 2006. Domain Decompostion of Stochastic PDEs and its Parallel. In: 20th International Symposium on High-Performance Computing in an Advanced Collaborative Environment (HPCS'06), pp. 17–21.
Shiginah, F.A.S.B. 2006. Multi-Layer Cellular DEVS Formalism for Faster Model Development and Simulator Efficiency. PhD thesis. Electrical and Computer Engineering Dept., University of Arizona.
Sipper, M., 1994. Non-uniform cellular automata: evolution in rule space and formation of complex structures. In: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems\Artificial Life IV, R.A., Brooks, Maes, P., (eds.), pp. 394–399.
Stephanov, A., Meng, L., 1995. The standard template library. HP Laboratories Technical Report 95-11 (R.1), November 14.
Stroustrup, B., 2000. The C++ Language, Special Edition. Addison Wesley.
Talia, D., 2000. Cellular processing tools for high-performance simulation. Computer, 44–52.
Tian, T., Burrage, K., 2005. Parallel Implementation of Stochastic Simulation for Large-scale Cellular Processes. In: Proceedings of the Eighth International Conference on High-Performance Computing in Asia–Pacific Region (HPCA-SIA'05) 0-7695-2486-9/05.
Von Neuman, J., 1966. Theory of Self-Reproducing Automata. University of Illinois Press.
Wainer, G., 2002. CD++: a toolkit to develop DEVS models. Software – Practice & Experience 32 (13).
Wainer, G., 2006. Applying Cell-DEVS Methodology for Modeling the Environment. Simulation, Transactions of the SCS 82 (10), 635–660.
Wainer, G., Giambiasi, N., 2001. Application of the Cell-DEVS paradigm for cell spaces modeling and simulation. Simulation 76 (1), 22–39.
Wolfram, S., 2002. A New Kind of Science. Wolfram Media.
Worsch, T., 1997. Programming environments for cellular automata. In: Proceedings Cellular Automata for Research and Industry (ACRI96), vol. 12. Springer, Berlin.
Wu, P., Wu, X., Wainer, G., 2004. Applying Cell-DEVS in 3D Free-Form Shape Modeling. In: Lecture Notes in Computer Science. Springer-Verlag, Heidelberg. 81–90.
Zeigler, B.P., 1976. Theory of Modeling and Simulation. Wiley, New York.
Zeigler, B.P., 1990. Object-Oriented Simulation with Hierarchical, Modular Models. Academic Press.
Zeigler, B.P., Praehofer, H., et al., 2000a. Theory of Modeling and Simulation. Academic Press.
Zeigler, B.P., Praehofer, H., Kim, T.G., 2000b. Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems. Academic Press.

Please cite this article in press as: Innocenti, E., et al., A software framework for fine grain parallelization of cellular models with OpenMP:..., Environ. Model. Softw. (2009), doi:10.1016/j.envsoft.2008.11.014