DESIGN OF A MULTITHREADED PARALLEL MODEL FOR FIRE SPREAD

Eric Innocenti
Alexandre Muzy
Antoine Aiello
Jean-François Santucci
University of Corsica
SPE – UMR CNRS 6134
B.P. 52, Campus Grossetti
20250 Corti. FRANCE.
e-mail : ino@univ-corse.fr

David R.C. HILL
ISIMA/LIMOS UMR CNRS 6158
Blaise Pascal University
Campus des Cézeaux BP 10125, 63177
Aubière Cedex France
e-mail : hill@isima.fr

KEYWORDS
Parallel simulation, fire modeling, DEVS formalism, multicomponent formalism.

ABSTRACT

We present an approach allowing the simulation of fire spread on parallel computers. The speedup obtained shows that the technique used is efficient. Our algorithm is based on the DEVS formalism for Discrete EVent Simulation. Two levels of abstraction are considered: a low and a high level. The low level takes into account particular conditions (vegetation, slope, wind, etc.) through cellular independent components which have their own states and behavior. The high level of abstraction considers an area of land with a fire front as a whole unit that evolves in time and space. Our design consists in proposing a multicomponent model. A set of active elements is defined and added to the multicomponent in order to improve the parallelism and to limit computations. We develop a two levels parallel approach. The first level, relying on fork() function calls, allows portable placement parallelism on real processors. The second level based on the parallelization of the active elements is adapted for hyperthreading processors, which authorize independent threads running at the same time. We use here POSIX thread library. The full advantage of all available CPUs and a significant speedup on shared memory multiprocessor machines are obtained. Experiments and results are commented on, in the last section.

1 INTRODUCTION

Computer simulation of fire spread involves spatial effects that remain a challenging problem in terms of computation time and memory, to the extent that we want to work with large propagation areas. In addition, it is important to provide decision-aid tools for fireman advisors, hence the model is based on physics in order to obtain an acceptable precision level [1].
In this paper, we expose a method able to simulate the fire propagation on a two-dimensional area. This modeling approach falls into the class of cellular propagation models [2]. Our aim is to predict the position of the fire-front, but with the help of physical equations and experimental parameters [3]. The Multicomponent approach, the DEVS formalism (Discrete EVent Simulation) and parallel computing provide us with a basis to tackle the simulation of large scale areas. This work is motivated by the following observations: accurate fire simulation based on physics is costly in computation time; taking into account large areas requires a considerable amount of memory not available on traditional computers; thus many simulation models depend on a parallel architecture.
Our answer to these issues consists in an original parallel fire model which reduces considerably the computations of cellular elements. In order to reach that objective, a propagation plan is decomposed into independent propagation domains which are computed in processes. Each one is linked to a list of cells which stores active component references, and thus we are able to limit the computations to elements that change their states. The set of active cells is distributed across different threads of the processors. Computation is then easily portable on parallel computers based on hyperthreading processors and is limited to the fire front. Moreover our model supports multiple simultaneous fire fronts. Experiments show the effectiveness of our approach both in terms of execution time and domain size.
Another contribution of this work is the introduction of a new fire spread approach, based on DEVS formalism combined with the multicomponent model, allowing two levels of abstraction. The parallel implementation of the algorithm uses the fork() calls and the POSIX threads library [4].

The paper is organized as follows. The next section recalls the concepts and the formalisms used. Section 3 describes in detail the simulation model developed. Section 4 explains the DEVS simulator implemented, the overall algorithm of the simulation is described precisely. Section 5 describes the testbed we used, including the results and discussion. Finally, the achievements and limitations are summarized and the directions of further investigations are given.

2. CONCEPTS AND FORMALISM

2.1. DEVS Formalism

Since the beginning of the 1970's, developing the theoretical basis on discrete event dynamic system modeling and simulation is an active research field.

DEVS is an abstract universal formalism used for discrete event modelling introduced by Bernard Zeigler. A basic DEVS model is a structure:

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

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 advance time function

The components of the model are described via the descriptive variables, S representing the subset of state variables. X is the subset of input variables and Y the subset of output variables. The atomic models are influenced by internal and external events. The external events are generated on one of the input ports of an atomic model; the internal events are programmed as a result of the external events and imply the model response to the outside. The atomic model activity is described by the internal transition function δ_{int}, by the output function λ and by the external transition function $\delta_{ext.}$. The reader interested in more details will benefit from the new reference book for DEVS [5].

2.2 Multicomponent Formalism

The modeling we consider has to take into account the structural diversity of the medium as well as the behavioral diversity of the various elements. In addition, the modelling choices should also facilitate multiple fire front and its parallelization. In order to reach these objectives, we use the multicomponent specification system introduced by Zeigler in [5].
A multicomponent is a structure:

$MC=<T, X, \Omega, Y, D, \{M_d\}>$,

where

T is a time base,
X is the input value set,
Ω is the set of allowable input segments,
Y is the output value set,
D is the set of component references.
For all d $\in$ D,

$$M_d=<Q_d, E_d, I_d, \Delta_d, \Lambda_d>$$

is a component with
Q_d, is the set of states of the component d,
E_d, is the set of its influencers,
I_d, is the set of its influencees,
Δ_d, is the state transition function of d,
Λ_d, is the output function of d.

Based on cellular automata, multicomponents are intrinsically parallel, thus they can be implemented efficiently onto parallel computers. In our case the communication flow between processors is low due to the regularity of the elements [6,7]. As stated previously the implementation of the parallel features of our algorithm will rely on fork and POSIX thread library functions [4].

3. FIRE PROPAGATION MODEL

3.1 Physical modeling

Before tackling the modeling, the phenomenon of the combustion of a solid is first specified. If a solid material is subjected to a quite important flow of heat, it will deteriorate: this is the chemical process of pyrolysis. Combustible gases are then delivered, the resulting flame comes from their reaction in stoechiometrical proportions, with the oxygen present in the atmosphere. The flame obtained is quasi isobaric and the reaction of combustion occurs in a narrow area which can be likened to a surface. The energy released by the pyrolysis is released in the atmosphere and also carried towards inert combustibles. Thus, these radiative and convective thermal transfers between the flame and the solid as well as the heat conduction will keep the fire spread going on.

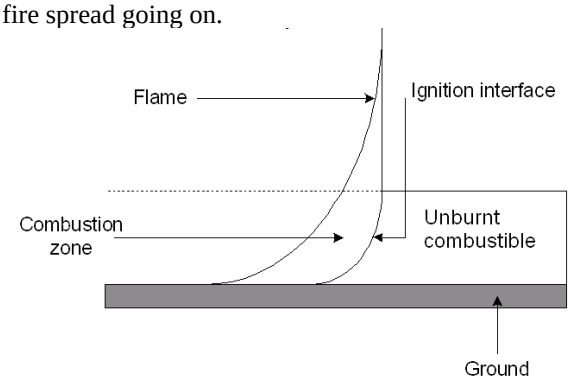

Figure 1: schematic representation of a fire front

We use as the basis for propagation behavior a one dimensional theoretical model, in which a second dimension can be obtained using propagation algorithms integrating empirically wind and slope [8]. Here, we consider fire spread within a 1 m² domain of pine needles, without slope nor wind. The spread plan is divided into elementary cells composing the ground and the plants. The previously physical study made by physicians of the University of Corsica allowed us to define a system of differential equations in order to describe the phenomenon [3].
In order to discretize the model, the method of finite elements is used so as to make its application easier. The domain considered is made up of 1 m² cells uniformly distributed and a 0.01 s time step is used. The resolution of the physical model, furnishes the following algebraic equation:

$$T_{i,j}^{k+1} = aT_{i-1,j}^{k} + aT_{i+1,j}^{k} + bT_{i,j-1}^{k} + bT_{i,j+1}^{k} + cQ\left(\frac{\partial \sigma_v}{\partial t}\right)_{i,j}^{k+1} + dT_{i,j}^{k} \quad (1)$$

where $T_{i,j}$ is the temperature of one cell of the domain.

The coefficients a,b,c,d depend on the time step and the size of cells. These coefficients are identified on the basis of the experimental data of temperature according to time. This equation represents the temperature curve of a cell of the domain, as shown in figure 2. Once the temperature T_{ig} is reached, the combustion of cell starts and finishes at temperature T_f.

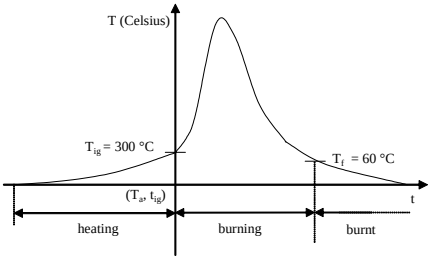

Figure 2: Simplified curve for the temperature of a cell

Since the modeling is based on the physical equation (1), it naturally makes us consider the spread domain as a cellular domain. The space is divided into a set of cells, each one with a temperature. Cell temperatures evolve through the simulation and each cell receives heat from its neighbors. A cell ignites when its temperature exceeds the T_{ig} threshold temperature. This approach will enable us to deal more easily with non homogeneous domains simply by changing the equation of an element. Consequently we are able to define different behaviors depending on of the elements of the domain.
In order to describe accurately the studied phenomenon, the multicomponent system specification described above is used because it is most suitable for the considered system.

3.2 Multicomponent Model

The atomic DEVS model that is envisaged is a multicomponent with an input port and an output port associated to a set of active cells. The input port starts the fire spread and the output port conveys the distribution of the temperatures of cells when the multicomponent has finished its internal transition (settlement of the temperatures). Two abstraction levels are considered: a high and a low level [9], the relation linking the two levels being a composition. At high level, the evolution of the front fire is governed by two transition functions; the external transition function δ_{ext} which updates the global state variables; the internal transition function δ_{int} which computes the active elements and updates the front fire. The temporary set needed for intermediate calculations is then reduced to a handful of elements.

The principal assets of our model are:

- the multicomponent approach which allows us to develop complex and accurate propagation models,
- the use of multicomponents coupled with a set of active cells which allows us to write easily parallel algorithms and take full advantage of shared memory multi-processor machines,
- the gain in time and memory requirements is susbtantial,
- the portability on parallel environment is facilitated.

At low level, a component is defined as being a specific address. A cell can access directly the state information of its neighbor cardinal cells. This approach allows each component to have its own set of states and a transition function. The internal transition relies on two key functions: **updateActiveSetFunction**() and **updatePropagationDomainFunction**(). The first one is responsible for updating the active cells and the second transfers the modifications on the multicomponent model using the addresses of the components. These two important stages of the internal transition involve many component function calls (Δ_d,Λ_d) that can be executed in parallel.

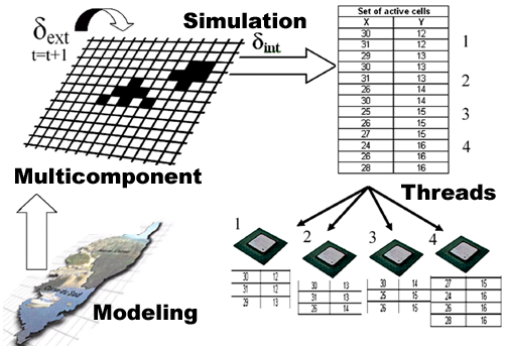

Figure 3: Schematic representation of the internal transition. The computation is limited to the fire- front, the set of active cells is partitioned in equal subsets and each one is computed into a thread

In a basic approach, the internal transition function δ_{int} had to pass through all the components [5]. This algorithm includes the execution of their individual output functions Λ_d, in order to define the output Y of the multicomponent, and their local state transition functions Δ_d, in order to update their states.
At component level, the first optimization consists in using a set of active cells (cells that must change their states at the next simulation time). The computation is then limited to the fire front components. Hence, this approach imposes the redefinition of the list of the active cells and for small domain sizes (less than 10 000 components in our case), component management generates overheads much greater than the original

approach. However, the overheads in the case of large scale domains are negligible.
The starting point of our second optimization is the set of active cells described above. This set is divided into equal subsets and each of them is placed into a thread. One thread has to perform the output and the transition functions of each cell of the subset and update the component linked to the multicomponent. During simulation the active cells are the most equitably packed. All the active cells of the multicomponent are updated synchronously; also we use synchronization routines to build simulation results. The evolution of the states of the active cells determines the global behavior.
At domain level, the optimization consists in decomposing the propagation area in smallest domains of propagation. Each one then constitutes an independent simulation process which is attached to a physical processor.
The disadvantages of the approach followed here are mainly memory and computation times overhead due to the large number of active components. Of course, this disadvantage is only present if a great part of the components are active, which is very improbable.
A less than optimal decomposition at high level generates bad active cells distribution that induces an imbalance in the workload distribution to the processors and requires some action, in order to rebalance the load over processors. The solution lies in the development of an efficient decomposition algorithm, which constitutes the next stage of our work.
The major advantages of this approach are the following: ability to compute multiple fire fronts, dynamic assignment of pack of components to threads, possibility to skip inactive components and last the solution retained gives faster simulation time both on sequential and parallel architectures. A transparent re-assignment of components to processors is achieved and the full advantages of hyperthreading processors, which authorize independent threads running at the same time, are exploited.
During the simulation, components burnt do not evolve any more. In this case, we remove the component from the set of the active elements, thus reducing computing and active list iterations. On the other hand, the course of the simulation will also require creation of additional active components due to the propagation of the front fire. Most generally, active components management is a combination of component removals on the one side and component adjustments on the other, augmented with packing and sending the cells and their states to another processes.

4. SIMULATION ALGORITHM

The kernel of our algorithm is a discrete event simulator based on the DEVS formalism. In order to apply this formalism to fire spread, two states are defined: active and passive. The overall propagation plan is divided into propagation domains. This one is active when fire spreads, passive if not. The propagation domain is considered as an atomic DEVS model for the simulator. We create as many processes as domains we considered. An event on their input port turns them active and they turn passive when they execute their internal transition. Each domain is attached to a process.

The algorithm can be divided into four steps:
1. Set the initial conditions for all model elements, and the initial bag of active cells.
2. Execute external transition.
3. Apply the internal transition function. We compute the next state S of the model and new cells (influencers) are added to it as it expands. The propagation domain is updated.
4. While the final simulation time is not reached, simulation back to step 3.

Events are represented through five messages which will enable us to put the simulation forward within an algorithm based on the DEVS formalism. The messages management will be made thanks to a schedule of dates, a clock guaranteeing the global time of the simulation as far as the root coordinator is concerned. The different types of exchanged messages will permit the pursuance of the simulation till final time t_f:

- (i,t)-message: is used to initialise the model with the group of rounded down values chosen by the user.
- (x,t)-message: this message is used when an external event occurs on one of the input ports of the model.
- (done,t)-message: is used to indicate that the model has completed its task; it's a message of settlement.
- (*,t)-message: indicates a state change of the model, due to an internal event.
- (y,t)-message: indicates the emission of an output event.

The atomic model is a multicomponent with input and output ports and a set of actives cells. The input port starts the fire spread and the output port conveys the distribution of the different temperatures of the cells when the multicomponent has finished its internal transition. As depicted in algorithm 1, the simulator uses two temporal variables: t_{last} and t_{next}. The first one serves to store the simulation time once the last event has occurred and the second one serves to store the scheduling time of the next event.

```
DEVS simulator
Variables
   tlast , tnext

DEVS    //→associated model

Switch (typeMessage)

   Case 'x' :
       If (tlast ≤ t ≤ tnext) Then
           e = t - tlast
           δext(x,t-message)
           tlast = t
           tnext = tlast+ta(s)
           send (done,t)-message
       Else
           Error« Bad synchronisation »
       End If
   Fin Case
```

```
   Case '*' :
       If t <> t_next Then
           Error« Bad synchronisation »
       Else
           Y = λ(S)
           send (y,t)-message
           δint()
           t_last = t
           t_next = t_last+ta(s)
           send (done,t)-message
       End If
   End Case
End Case

End DEVS Simulator
```

Algorithm 1: DEVS simulator

The next event time t_{next} is sent to the parent coordinator in order to permit a good synchronization of the events. The root coordinator implements the loop dedicated to the whole simulation. It distributes the tasks corresponding to the events scheduled to its direct subordinate processors. The simulation length is easily computed from the t_{final} time which is required from the user; the root coordinator simulates the propagation until the t_{final} time is reached.

```
DEVS root coordinator
Variables
     t_sim   //→ current time of simulation
     t_final //→ final time of simulation
Simulator //→direct subordinate simulator

While ((Scheduler Not Empty) and (t_final<>t))
   //→reading   the   first   scheduler   message
   Scheduler.ReadMessage()

   Switch (typeMessage)

       Case 'done' :
           t_sim = t
           If (t_sim < t_final)Then
                send (*,t)-message
           End IF
       End Case

       Case 'y' :
           Save_State()
       End Case

       Case 'i' :
           // the model will evolve
           // until
           // time t of(i,t)- message
           t_final = t
           //model initialisation
           init_Model()
           //Start simulation
           send(x,t0)-message
       End Case

   End Switch

End While

End DEVS root Coordinator
```

Algorithm 2: DEVS root coordinator

We run the simulation scheme, described above, on the different processors of the parallel computer.

5. RESULTS AND COMMENTS

We show here the experimental results by using the approach presented above. The experiment consists in a homogenous multiple point lighting. To simplify the analysis of the results, the burnt space is decomposed in equal propagation domains. Each domain consists in a multicomponent where each component in the matrix represents a square area of land.

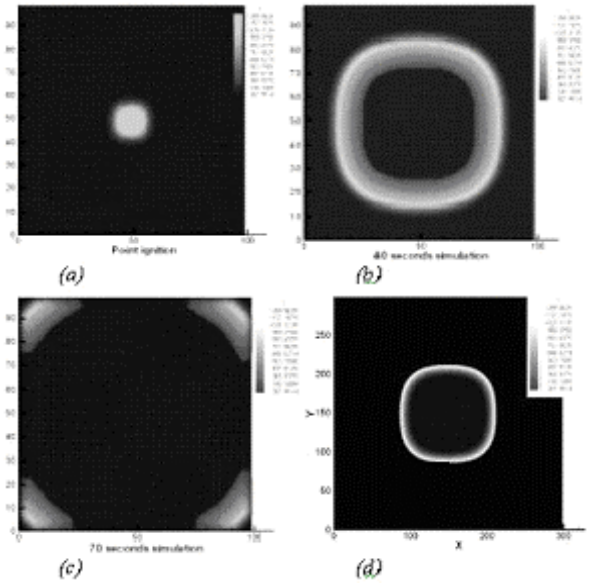

Figure 4: Four snapshots of the fire spread simulation in a multicomponent

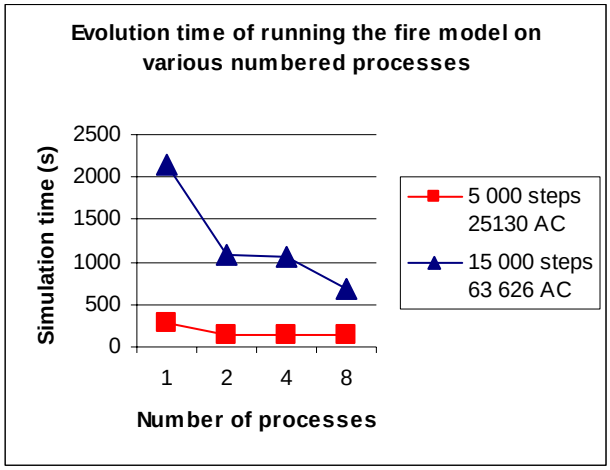

Figure 5: Simulation time for two experiments of simulation on various numbered processes. The maxima of active components reached are indicated

The measurements were obtained running on a bi-processor Intel Pentium 2.4 Ghz XEON distributed memory computer. We measured the time cost of running the fire model on various numbered processes and computed the speedups that we obtained.

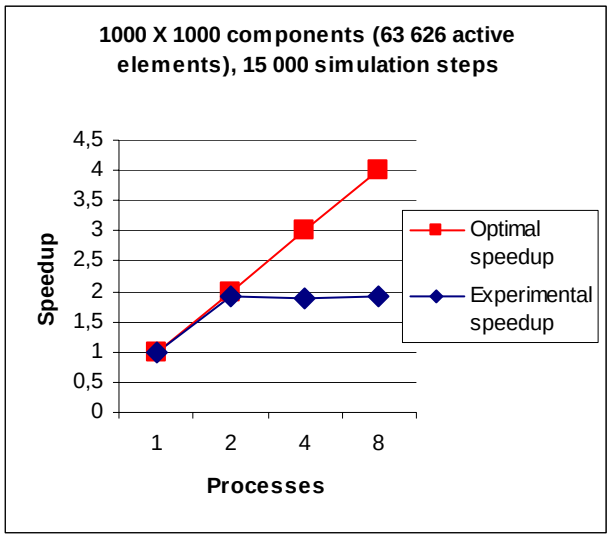

Figure 6: Speedup obtained for 150 s simulation time on a DELL XEON Pentium biprocessors

Simulation times are not directly dependent on the size of the domain, they depend on the number of active cells and processors, and the results showed in figure 6, compare them with the line y = x which we define to be the optimal speedup.

6. CONCLUSION

We have developed a new technique for simulating fire propagation on large domains with DEVS formalism, multicomponent modeling and parallel computation. An efficient model and a solution optimized for parallel simulation of fire spread are presented. The fire propagation is based on physical laws and a set of active cells is used to manage the fire-front. The DEVS simulator is based on a lattice of cells described as a multicomponent. This model can be applied to large areas and more complex configurations. The use of parallel techniques allowed us to obtain satisfactory computation time. In addition, the use of the multicomponent approach allowed precise description of the behavior of each cell. Our model also supports multiple simultaneous fire fronts. This approach does not require any special training in parallel computing from the end-user. The parallel simulation framework introduced, intended to simplify the parallelization of complex simulation models based on a lattice of components. The practical aspect lies in the fact that parallelization difficulties are hidden by the model, enabling an efficient and accurate simulation exploiting computational concurrency at a minimum cost. In order to increase the number of processors, that are generally limited on SMP machines, we plan to work on an adaptation of this algorithm for the execution through a cluster of workstations.

7. REFERENCES

[1] P. Eklund, S. Kirkby, J. Mann. 1999. "A Distributed Spatial Architecture for Bush Fire Simulation", Transactions on GIS, Blackwell Publishers Oxford, Vol. 3, No 3.

[2] M. S. Veach, P. Coddington, G.C. Fox. 1994. "BURN: A Simulation of Forest Fire Propagation.". Available online from http://citeseer.nj.nec.com/cs.

[3] J.H. Balbi and P.A. Santoni. 1998. "Dynamic modelling of fire spread across a fuel bed", Int. J. Wildland Fire, pp. 275-284.

[4] F. Mueller. 1993. "A Library Implementation of POSIX Threads under UNIX" in Proceedings of the USENIX Conference, Jan, pp. 29-41.

[5] B.P. Zeigler, H. Praehofer and T.G. Kim. 2000. "Theory of modelling and simulation", 2nd Edition, Academic Press.

[6] D. Talia. 2000. "Solving Problems on Parallel Computers by Cellular Programming", IPDPS Workshops,Springer Verlag Heidelberg, pp. 595-603.

[7] G. Spezzano, D. Talia, 1999. " Programming cellular automata algorithms on parallel computers" in Future Generation Computer Systems, No 16, pp. 203-216.

[8] E. Pastor, L. Zàrate, E. Planas, J. Arnaldos. 2003. "Mathematical models and calculation systems for the study of wildland fire behaviour", Progress in Energy and Combustion Science, No 29, pp. 139-153.

[9] J. Jorba, T. Margalef, E. Luque, J.C.S. Andre, D.X. Viegas. 1999. "Parallel Approach to the Simulation of Forest Fire Propagation", Umwelt-informatik-zwischen Theorie und Industrie-anwendung Umweltinformatik' 99. Metropolis-Verlag, Germany, pp. 69-81.