Laser Dynamics Modelling and Simulation: An application of Dynamic Load Balancing of Parallel Cellular Automata

J.L. Guisado, F. Jiménez-Morales, J.M. Guerra, F. Fernández de Vega, K.A. Iskra, P.M.A. Sloot, and Daniel Lombraña González

1 Introduction

This chapter reviews the application of a biologically inspired heuristic technique – Cellular Automata (CA) – for developing high performance simulations of a well known complex system: the laser.

CA can be described as a class of mathematical systems. They were introduced several decades ago, and are well suited to model spatio-temporal phenomena. On

J.L. Guisado
Departamento de Arquitectura y Tecnología de Computadores, Universidad de Sevilla.
E.T.S. Ingeniería Informática. Avda. Reina Mercedes s/n. 41012 Sevilla, Spain.
e-mail: jlguisado@us.es

F. Jiménez-Morales
Departamento de Física de la Materia Condensada, Universidad de Sevilla.
P.O. Box 1065, 41080 Sevilla, Spain. e-mail: jimenez@us.es

J.M. Guerra
Departamento de Óptica, Facultad de CC. Físicas, Universidad Complutense de Madrid.
28040 Madrid, Spain. e-mail: jmguerra@fis.ucm.es

F. Fernández de Vega
Centro Universitario de Mérida, Universidad de Extremadura.
Sta. Teresa Jornet, 38. 06800 Mérida (Badajoz), Spain. e-mail: fcofdez@unex.es

K.A. Iskra
Argonne National Laboratory, Mathematics and Computer Science Division.
9700 South Cass Avenue, Argonne, IL 60439, USA. e-mail: iskra@mcs.anl.gov

P.M.A. Sloot
Section Computational Science. Laboratory for Computing, System Architecture and Programming. Faculty of Science, University of Amsterdam.
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands. e-mail: p.m.a.sloot@uva.nl

D. Lombraña González
Centro Universitario de Mérida, Universidad de Extremadura.
Sta. Teresa Jornet, 38. 06800 Mérida (Badajoz), Spain. e-mail: daniellg@unex.es

the other hand, CA can be implemented very efficiently on parallel platforms, given both, their intrinsic parallel nature, with all the components working usually in a synchronized way, and the discreteness of the individual components using the same behavior rules. We therefore make use of this feature, and consider the problem of running Parallel CA simulations on non-dedicated clusters of workstations. We thus present results of laser dynamics simulations, traditionally modeled using differential equations.

This new approach can be very useful when modeling lasers given that differential equations are difficult to integrate or even difficult to apply: lasers ruled by stiff differential equations, devices with complex boundary conditions, very small devices for which the approximations implied by the differential equations may not be valid, etc.

The presented model is based on a synchronous CA using the Single Program, Multiple Data (SPMD) paradigm, deployed on a non-dedicated cluster of computers. Therefore, it is not clear in advance if a good performance and efficiency can be obtained on this kind of non-dedicated platform. We thus analyze the feasibility of executing our parallel bioinspired model of laser dynamics on an heterogeneous non-dedicated cluster, and we evaluate its performance including artificial load to simulate other tasks or jobs submitted by other users. Finally, a dynamic load balancing strategy is used with two main differences from previous CA implementations:

- It is possible to migrate the load to cluster nodes that do not belong initially to the pool.
- The model uses the load balancing tool –the Dynamite system– to give flexibility to the model.

By studying the performance and scalability of this parallel implementation we obtain very satisfactory results, including performance increases from 60% to 80%.

This chapter is organized as follows: In Section 2, we will present the problem to be solved by the proposed algorithm: laser dynamics. We will describe in detail the proposed algorithm in Section 3. In Section 4, we will review some of the laser properties which are successfully reproduced by the CA model. Next, we will describe a parallel implementation of the CA model and we will analyze its performance and scalability when executed on a heterogeneous non-dedicated cluster, including dynamic load balancing. In Section 6, we will propose some ideas for future work. Finally, some conclusions will be drawn in Section 7.

2 The Problem: Laser Dynamics

A laser is a device that generates and amplifies coherent electromagnetic radiation based on the stimulated emission phenomenon, predicted by Albert Einstein in 1917 [1]. The word laser is an abbreviation of “Light Amplification by Stimulated Emission of Radiation”. In a laser system radiation is amplified by propagation across a

medium, in which, the population of an upper energy state is larger than the population of a low energy state (population inversion). Some mechanism, usually known as the pumping system, is needed to enhance the upper state population up to be larger than the remaining in a lower energy state. When the pumping is above a threshold value, the radiation traveling through the medium is amplified by the stimulated emission process. The effective amplification is usually enhanced by placing the laser active medium inside a Fabry-Perot resonator, that provides a feed-back by making the amplified light bounce between two parallel mirrors. Therefore, the laser device acts as a regenerative light oscillator and transient, periodic or chaotic oscillatory processes can be originated in it.

The time dependence of the total number of laser photons and the total population inversion in the laser medium can be described [2], as a first step, by the Equations (1) and (2):

$$\frac{dn(t)}{dt} = KN(t)n(t) - \frac{n(t)}{\tau_c} \tag{1}$$

$$\frac{dN(t)}{dt} = R - \frac{N(t)}{\tau_a} - KN(t)n(t) \tag{2}$$

This model, based on two coupled nonlinear rate equations is simplified but can still describe realistically many laser dynamics phenomena. The first equation provides the variation on the number of laser photons $n(t)$ with time, proportional to the laser beam intensity. The term $+KN(t)n(t)$ describes the increase in the number of photons by stimulated emission (K is the coupling constant between the radiation and the population inversion). The term $-n(t)/\tau_c$ accounts for the decaying (or absorption) process of laser photons inside the laser cavity with a characteristic decay time τ_c. The second equation represents the temporal variation of the population inversion $N(t)$. The term $+R(t)$ represents the pumping of electrons with a pumping rate R to the upper laser level. The term $-N(t)/\tau_a$ introduces the decaying of electrons from the upper laser level to lower levels, with a characteristic decay time τ_a. The product term $-KN(t)n(t)$ reflects the decreasing of the population inversion by stimulated emission. The presence of the product term $KN(t)n(t)$ in each equation gives them a nonlinear nature. For small amplitude fluctuations its solutions can show relaxation oscillations in their evolution towards a steady state. For strong oscillations the two variables $n(t)$ and $N(t)$ are changing in a fast and typically nonlinear way and there does not seem to be a simple analytic solution [2, 3].

The four-level laser system shown in Figure 1 is a simplified model that still gives a realistic description of the main phenomena featured by a laser system: for instance, an external pumping process can excite electrons and make them jump from the ground level up to level E_3. Similarly, the figure shows the population inversion process, produced between levels E_1 and E_2 thanks to the fact that the life times of energy levels E_3 and E_1 are negligible as compared to the life time of level E_2. Therefore, electrons in levels E_3 and E_1 decay very fast but level E_2

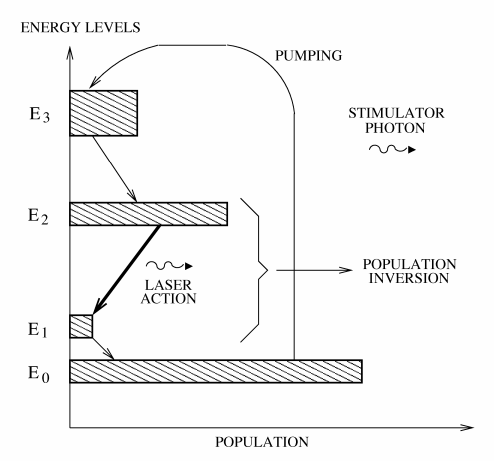

Fig. 1 A four-level laser system and its basic physical processes.

is metastable. On the other hand stimulated emission occurs when an electron in level E_2 decays down to level E_1 stimulated by the presence of a stimulator photon with energy $E = E_2 - E_1$. In addition, there are two processes which are also very important and which are not represented in Figure1: absorption of electrons in level E_2 (which decay to lower levels due to different processes not related to stimulated emission) and absorption of laser photons, a fraction of which disappear because they leave the laser cavity through the semi-reflecting mirror or are absorbed by the material.

3 Cellular Automata and Laser Dynamics

Cellular Automata (CA) are a class of spatially and temporally discrete mathematical systems characterized by local interaction and synchronous dynamical evolution [4]. They provide an excellent approach for modeling and simulating complex systems and have been used over the recent years in many fields of science and technology [5, 6]. We study here the modeling of light amplification by stimulated emission by means of CA, firstly described by Guisado et. al. in [3].

The algorithm is based on a two-dimensional, partially probabilistic, multivariable CA that simulates a transverse section of the active medium in a laser system. The defining characteristics of the CA are described in the following.

3.1 Cellular Space

The CA employ a cellular space consisting of a two-dimensional square lattice which contains $N_c = L \times L$ cells. Periodic boundary conditions are used.

3.2 State of the Cells

Each of the cells within the CA embodies two variables: $a_{ij}(t)$ and $c_{ij}(t)$. The first one, $a_{ij}(t)$, represents the state of the electron in cell $\{ij\}$ (row i and column j) at time t: when $a_{ij}(t) = 0$ the electron is in the ground state and when $a_{ij}(t) = 1$ the electron is in the upper laser state. Also, $c_{ij}(t) \in \{0, 1, 2, ..., M\}$ represents the number of laser photons in cell $\{ij\}$ at time t. This number is bounded by an upper value M which must be chosen large enough to avoid the saturation of the system. The state variables represent "bunches" of real photons and electrons. Their values are linked to the number of photons and electrons in the real system by a normalization constant.

3.3 Neighborhood

Every cell performs local interactions with a predefined number of surrounding cells. We employ the well-known *Moore neighborhood* for establishing interactions patterns: the neighborhood of any particular cell is formed by the cell itself (C), its four nearest neighbors located at the north (N), south (S), east (E) and west (W) positions and the four next nearest neighbors located at the northeast (NE), southeast (SE), southwest (SW) and northwest (NW) positions, as shown in Figure 2 .

NW	N	NE
W	C	E
SW	S	SE

Fig. 2 Moore neighborhood.

3.4 Transition rules

Every CA model requires a set of rules which defines the behavior and evolution of the whole system. This set of rules, usually known as transition rules, specify the state of each cell at time step $t+1$ depending on its state and the state of the cells included in its neighborhood at time step t. Therefore, the rules model the physical processes working at the microscopic level in the laser system. The application of the transition rules is the main operation of a CA algorithm. In our case the overall structure of the CA laser model algorithm is shown in Algorithm 1.

Algorithm 1 Pseudo-code diagram for the CA laser model

```
Initialize system
Input data
for time step = 1 to maximum time step do
   for each cell in the array do
      Apply stimulated emission rule
      Apply photon decay, electron decay, and pumping rules
      Apply noise photons creation rule
   end for
   Calculate populations after this time step
   Optional additional calculations on intermediate results
end for
Final calculations
Output results
```

After an initialization step, the transition rules are applied synchronously to each CA cell inside a time loop. Our CA model employs five transition rules:

- Stimulated emission rule: If the electronic state of a cell has a value of $a_{ij}(t) = 1$ at time t and the sum of the values of the laser photons states in the nine neighboring cells is larger than a certain threshold θ (which in our simulations has been taken to be 1), then at time $t+1$ a new photon will be emitted in that cell: $c_{ij}(t+1) = c_{ij}(t)+1$ and the electron will decay to the ground level: $a_{ij}(t+1) = 0$. All the cells of the CA must be updated in parallel. To this end, changes from this rule are computed using a temporal matrix c'_{ij}. After the rule has been applied to all the cells of the CA, the values of c_{ij} are updated with the contents of c'_{ij}.
- Photon decay rule: Each photon is destroyed τ_c time steps after being created. In particular, (tlc_{ijk}) represents the number of time steps that will have to elapse until a particular photon located in cell $\{ij\}$ (at row i and column j) is destroyed, where k distinguishes between the different photons that can occupy the same cell. When a photon is created, $tlc_{ijk} = \tau_c$. After that, 1 is subtracted from tlc_{ijk} at each time step and the photon will be destroyed when $tlc_{ijk} = 0$.
- Electron decay rule: After an electron is excited from the ground level to the upper laser level, it will decay to the ground level again after τ_a time steps, if it has not yet decayed by stimulated emission. In particular, (tla_{ij}) represents the number of time steps that will have to elapse until a particular electron located

in cell $\{ij\}$ decays to the ground level. When the electron is initially excited, $tla_{ij} = \tau_a$. After that, 1 is subtracted from tla_{ij} at each time step and the electron will decay to the ground level again when $tla_{ij} = 0$.

- Pumping rule: If the electronic state of a cell $\{ij\}$ has a value of $a_{ij}(t) = 0$ at time t, then at time $t+1$ that state will have a value of $a_{ij}(t+1) = 1$ with a probability pumping λ.
- Noise photons creation: A small number of laser photons in randomly chosen positions is introduced at each time step to reproduce spontaneous emission and thermal contributions, responsible of the initial laser start-up. To this end, for a small number of randomly chosen cells $\{ij\}$ ($< 0.01\%$ of total) it is applied $c_{ij}(t+1) = c_{ij}(t) + 1$.

This CA models a typical four levels laser system, as described in section 2 and represented by the diagram shown in Figure 1. Thus it has been assumed that the population of the lower laser level (level 1 in Figure 1) is negligible. For this reason, stimulated absorption transitions have not been taken into account. Also for the same reason, the state $a_{ij}(t) = 0$ doesn't correspond to the lower laser level, but to the ground level, which is coupled to the pumping mechanism.

4 Experimental Analysis

We first present a summary of experimental results obtained when applying the previously described model to the simulation of laser dynamics (for interested readers, a whole description of results can be found in [3, 7, 8, 9]). We analyze in this section how the CA model of laser dynamics can reproduce different aspects of the phenomenology of laser systems.

Three main parameters influence the behavior of the system: the pumping probability (λ), the life time of photons (τ_c) and the life time of excited electrons (τ_a). In a simulation, an initial state is provided ($a_{ij}(0) = 0$, $c_{ij}(0) = 0$, $\forall ij$, except a small fraction, 0.01%, of noise photons present) and then the system is let to evolve for a number of time steps. In each step, we measure two macroscopic magnitudes: the total number of laser photons, $n(t)$, and the total number of electrons in the upper laser state (population inversion), $N(t)$, defined in Equation 3.

$$n(t) = \sum_{i=1}^{L_x} \sum_{j=1}^{L_y} c_{ij}(t), \qquad N(t) = \sum_{i=1}^{L_x} \sum_{j=1}^{L_y} a_{ij}(t) \tag{3}$$

It is a well-known laser systems' feature that laser action only happens when the pumping probability is over a threshold value. This property is correctly reproduced by the CA model [3] and the dependence of this threshold value on the other two system parameters (life times τ_a and τ_c) is found to be in good agreement with the laser behavior, as shown in Figure 3.

Two different behaviors which depend on the values of their three main parameters can be found in their time evolution: a constant or an oscillatory behavior [3, 8].

As shown in Figure 4, the model reproduces these two kinds of behavior: the time evolution obtained from the simulations is similar to that one exhibited by laser systems, described for example in [2]. A lattice size of 400×400 cells was used for this figure.

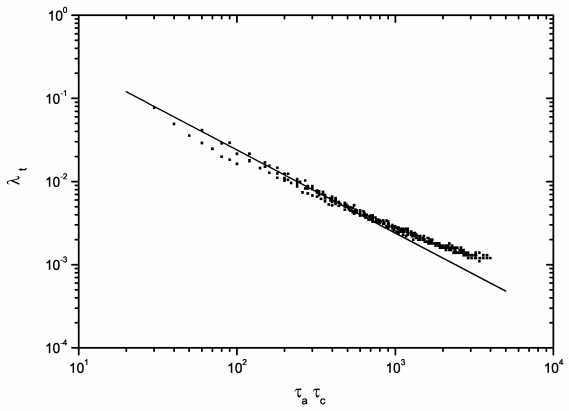

Fig. 3 Dependence of the threshold pumping probability λ_t from the CA laser model on the product of the characteristic life times τ_a and τ_c (measured in time steps), plotted on a logarithmic scale. The solid line is the laser behavior predicted by the standard laser rate equations, and the dots are the results of the simulations.

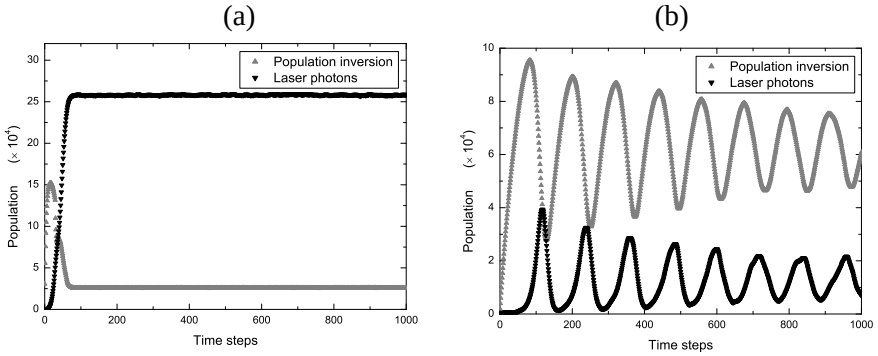

Fig. 4 Results of the simulations, showing the time evolution of the two macroscopic magnitudes —number of laser photons $n(t)$ and population inversion $N(t)$— versus time, for two different sets of values of the system parameters. The two main characteristic behaviors exhibited by lasers are reproduced by the CA model: (a) (left): Constant behavior. Parameters: $\{\lambda = 0.192,\ \tau_c = 10,\ \tau_a = 30\}$. (b) (right): Oscillatory behavior. Parameters: $\{\lambda = 0.0125,\ \tau_c = 10,\ \tau_a = 180\}$.

Moreover, we can also notice in Figure 5 another complex behavior in the CA model: irregular oscillations with fluctuations on a wide range of time scales appear

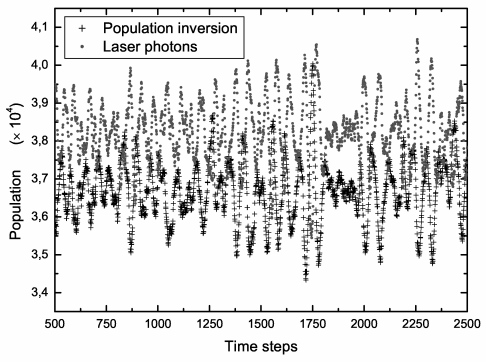

Fig. 5 Regime with irregular oscillations, for: $\lambda = 0.031$, $\tau_c = 10$, $\tau_a = 180$. The number of laser photons and population inversion are plotted versus time, after a transient of 500 time steps. Lattice size: 400×400 cells.

(see [8]). This regime could correspond to a chaotic state, as found in the dynamics of many lasers. Also, the dependence on the system parameters of the type of behavior exhibited in the time evolution of the system is in a good qualitative agreement with the laser behavior [3], as shown in Figure 6.

In this figure, we show a Contour plot of a magnitude called the *Shannon's entropy* of the distribution of the number of laser photons, for a fixed value of $\tau_c = 10$ time steps and obtained using simulations with a 200×200 lattice. This magnitude is a good indicator of the presence of oscillations in the time evolution of the number of laser photons (for a precise definition and discussion, see for example [7]). In this plot, R is the laser pumping rate and R_t is the threshold laser pumping rate, which are linearly related to the pumping probability λ and the threshold pumping probability λ_t that appear in the CA model, so that $\frac{R}{R_t} = \frac{\lambda}{\lambda_t}$. Points a, b and c show the values of the parameters that correspond to Figs. 4 and 5: a corresponds to a constant behavior (Figure 4 left), b to a oscillatory behavior (Figure 4 right), and c to a regime with irregular oscillations (Figure 5).

High values of the Shannon's entropy (dark zones) correspond to an oscillatory behavior and low values (bright zones) to a non-oscillatory response. The predictions of the standard laser rate equations are indicated by the black line: areas of oscillatory behavior should appear above and to the right of this curve and constant behavior should appear in the remaining areas. There is a good qualitative agreement between the predictions and the results of the simulations indicated by the Shannon's entropy, as the high values of this magnitude appear above and to the right of the black line and their contour resemble the shape of this line.

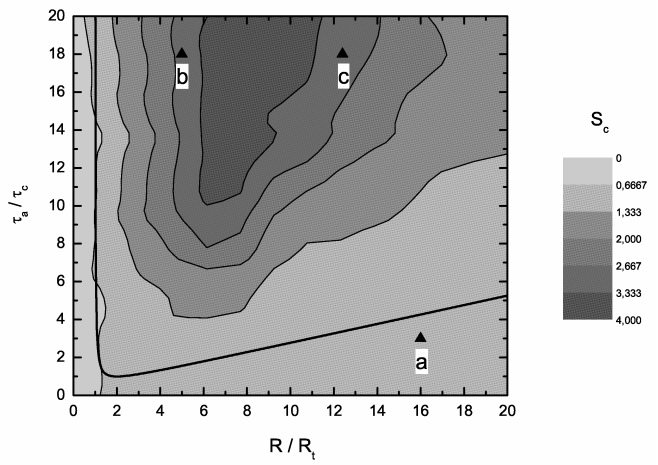

Fig. 6 Contour plot of the Shannon's entropy of the distribution of the number of laser photons obtained from the simulations with a fixed value of $\tau_c = 10$ time steps. This plot shows there is a good qualitative agreement between the dependence on the system parameters of the type of behavior exhibited by the system, as obtained from the simulations, and the laser behavior, delimited by the black line.

5 Parallel CA based simulation of the Laser

As shown in previous section, the CA based laser model correctly reproduces much of the phenomenology of the laser system, and can be therefore considered an alternative to the standard modeling approach, which employs differential equations. Even when such a very simple coarse-grained CA model has demonstrated its usefulness, if we pursue more realistic simulations for specific laser devices, and we want a larger granularity, closer to real macroscopic systems, a 3D CA – or huge 2D CA instead– may be required. Therefore, a very large lattice size will be needed, which will make necessary parallel computers systems to run the model, avoiding thus the otherwise prohibitively large runtime of sequential counterparts.

In this section, we describe the parallel implementation of the previous CA model and study its performance and scalability running on a small computer cluster (interested readers can refer to [9], [10], [11] and [12] for a larger description of results obtained). We begin by reviewing previous approaches to parallel implementations of CA models.

5.1 Previous approaches

As described above, sequential CA-based simulations can only be used for very simple systems. In order to simulate real world phenomena (which need 3D or large

2D CA) parallel implementations running on high performance parallel computers must be used since very long computing time or memory requirements are needed [13].

During the last decade some attempts to introduce parallelism within CA have been described. Most of them were not intended to implement in a direct way the inherently parallel CA internal rules of working, which can be easily simulated in a sequential fashion, but to improve speedup of the whole process by using a large number of processors.

The first attempts to parallelize CA were carried out by M. Resnick with the StarLogo system [14] and by Cannataro et. al. [15], although many approaches and results have been described later by using parallel CA, such as CAMEL [16], Nemo [17], PECANS [18], DEVS [19] and P-CAM [20]. A review of the topic is presented by Talia in [13].

Two main kinds of hardware infrastructure can be found in the literature for implementing parallel CA. The first one consists of using parallel computers. The second one requires specialized hardware such as the Cellular Automata Machine (CAM) [21]. In this work, we focus on using available parallel, cluster or GRID deployments. In fact, general-purpose parallel computers are well-suited for scalable CA models, from the point of view of speedup, programmability and portability.

Considering the structure of CA, the parallel implementation must take care of the information about the state of the cells included in the borders of the different partitions of the system; this information must be exchanged after each time step, as represented in Fig. 7. Therefore some methodology is required for the implementation of the parallel CA that allows information exchange among the different processes involved. Two main solutions are available: using general purpose parallel programming languages, such as HPF, HPC++ or Linda, or employing a standard high-level sequential language combined with specific libraries allowing parallel applications to run, such as MPI (Message Passing Interface), PVM (Parallel Virtual Machine) or OpenMP (Open Multi Processing).

On the other hand, when considering the information that must be exchanged among the different processes implementing the parallel CA after each time step, all of them must wait until all the computing nodes have finished for each time step before proceeding, i.e. the system operates in a lock-step mode. Therefore, the performance of the parallel implementation is limited by the slowest running task. A group of overloaded nodes which execute the computations slower than the majority of the nodes can degrade the overall performance. As the usual platform for executing this kind of applications are non-dedicated (and often heterogeneous) clusters, it raises the following question: Can this algorithm have a reasonably good performance when running on such platforms?

Several proposals in the literature focus on distributing the active cells between the nodes for CA featuring some cells that may become idle for a number of time steps [22, 15, 23]. Similarly, the possibility of moving cells from heavily loaded nodes to more unloaded ones has been described in [24, 25, 26]. A different possibility is to adjust the size of the partitions to be handled by each cluster node (see for instance [27, 28]). The idea behind those proposals is to make a balance on the

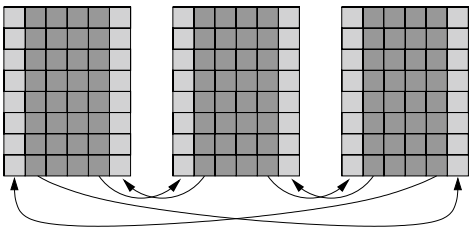

Fig. 7 In the parallel implementation of a CA, information of the state of the cells included in the borders of each partition of the system has to be communicated to the neighboring partition to be used in the computation corresponding to the next time step. In this example, the CA has been partitioned into parallel stripes. Each partition is assigned to a different processing node.

load when some nodes are overloaded. Yet, all of these proposals lack the capability of migrating jobs to new nodes which are dynamically added to the pool of computers initially running the CA model. This possibility, if present, would provide extra flexibility for a real-life non-dedicated parallel computing environment. This is the main reason why we opted for using a dynamic load balancing approach.

Even when load balancing has already been considered for parallel CA, most of the approaches include the dynamic load balancing mechanism within the CA algorithm (check for instance the P-CAM system [20] which directly integrates data decomposition and dynamic load balancing into the framework functionality). Our proposal tries to split and differentiate both important but non-related sections of the algorithm, the parallel CA and the load balancing technique. This would allow in the future to transparently change or improve any of them without affecting each other.

The tool that we have employed is *Dynamite* [29], an automated load balancing system that can migrate individual tasks which are part of a parallel program running with a message passing library. Dynamite is based on Dynamic PVM [30], a re-implementation of the PVM message passing library that includes the load balancing functionality. It monitors the utilization of the cluster nodes and migrates tasks when some of them get under-utilized or over-utilized as defined by configurable thresholds.

The Dynamite system is composed of three separate parts (see [31] and [32] for a complete description): the load-monitoring subsystem, the scheduler –which determines when a migration becomes necessary, which tasks will be involved and which particular allocation will be adopted– and the task migration software.

We have chosen Dynamite because of its flexibility, maturity, and availability. Nevertheless, dynamite is not the only load balancing system available. Other alternatives could also be used to execute this kind of simulations, such as the CAMELot-Grid system [33], and also the general purpose framework designed by Vadhiyar and Dongarra, implemented and tested in the GrADS system [34]. Even when they have some advantages over Dynamite, such as their possibility of integration on a grid

computing environment, we preferred Dynamite. The main reason was our interest for running the experiments on a cluster computing environment –mainly because of the tightly coupled nature of the parallel CA model– : a parallel CA (a *high performance computing* application) requires low latency of the communications, and this cannot be generally attained on a GRID environment (which generally would be more adequate for running multiple executions of a complete CA for different values of the parameters –a *high throughput computing* application–).

5.2 Algorithm description: Basic approach

In order to parallelize the CA model, and taking into account all the considerations included above, we decided to employ the message passing paradigm. Given the selection of Dynamite as a component for implementing and analyzing dynamic load balancing mechanisms, we selected the *parallel virtual machine (PVM)* implementation of this paradigm. A master-worker model was implemented, such as the one described in Figure 8.

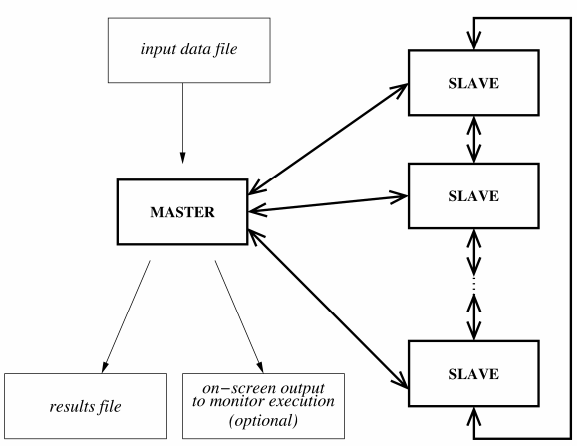

Fig. 8 Block diagram of the parallel implementation of the CA model of laser dynamics, showing processes running on different processors (boxes in bold type represent different processors), communications between them (bold lines) and data flows.

The *data decomposition methodology* is employed for distributing identical tasks with different partitions –of equal size– of the data among the pool of computers employed for the simulation, one partition per processor. Master and worker programs are therefore in charge of the following tasks:

- Master program:

1. Read input data (system size, number of partitions, parameter values, number of time steps) and initialize.
2. Spawning of slave programs.
3. Partitioning of the initial data of the automaton.
4. Sending of common information and initial data to each slave.
5. Collection of results from slaves at each time step.
6. Termination of slave programs.
7. Calculations performed using collected data.
8. Output of final data to external files.
9. Timing functions to measure performance.

- Slave program:

1. Reception of common information and initial data from master.
2. Time evolution computation for the assigned partition: application of CA evolution rules.
3. Exchange of state of the boundary cells with slave programs computing the neighboring partitions.
4. Computation of intermediate results and their communication to master program.

Among the possibilities for establishing the domain decomposition (see [9]), a 1D decomposition has been used, so that the CA is divided into parallel vertical stripes, each of them assigned to a different computer. Extra ghost cells have been included both at the left and right sides of each partition (see Fig. 7) which are in charge of storing the state of border cells belonging to adjacent partition, and allows to compute all the transition rules for the cells. The state information required from neighboring cells is the photon state $c_{ij}(t)$, which will be sent from neighboring subdomains and stored in the ghost cells. Each slave program is responsible for computing the time evolution on its assigned partition.

At the beginning of each iteration the state of the boundary cells is directly exchanged between slave programs computing neighboring partitions, using two couples of PVM send and receive routines (`pvm_send` and `pvm_recv`). The routine `pvm_recv` is blocking, so it waits until the specified message has arrived. Therefore, this exchange plays the role of a synchronization point between all the slave programs. This is illustrated in Fig. 9 which shows a detail of the tasks executed by each node and the messages transferred between different nodes versus time, once the computation has started. This figure also shows that computation periods are much longer than communication periods, so that the application achieves a high computation-to-communication ratio, of the order of 10.

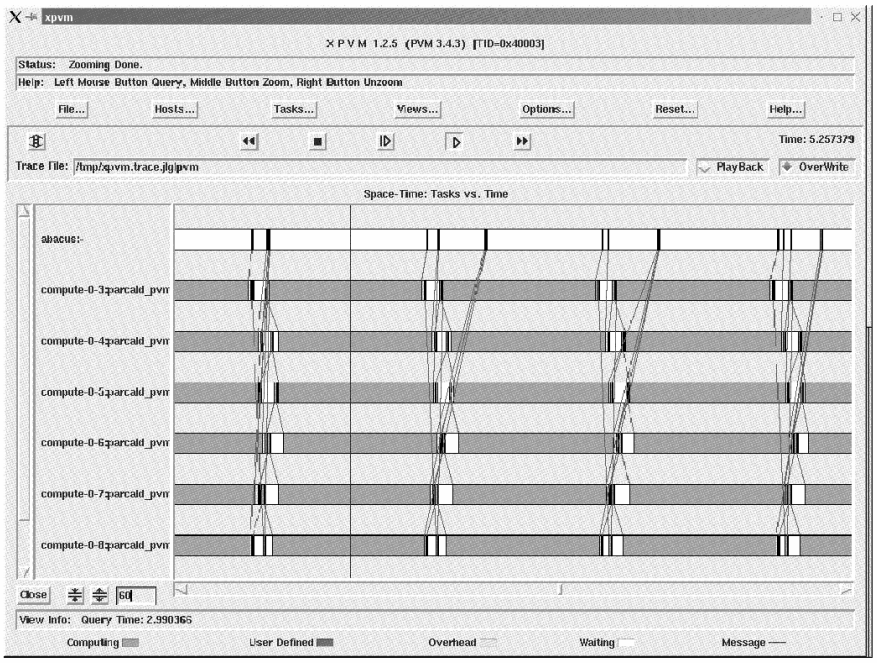

Fig. 9 Gantt chart depicting a detail of the tasks executed by each cluster node and the messages transferred between different nodes versus time, once the calculation has started. The exchange of neighboring states between nodes processing adjacent partitions at the beginning of each iteration acts as a synchronization point.

5.3 Performance and Scalability Analysis

We have measured the performance and scalability of the parallel CA by running simulations on the Beowulf-type cluster "Abacus" from the University of Extremadura (Spain), see Table 1.

Table 1 Abacus hardware specifications

Nodes	10
Microprocessor	Pentium-4
Clock Frequency	2.7 GHz (6 nodes) and 1.8 GHz (4 nodes)
Network	100 Mbps Fast-Ethernet switch

To avoid indeterminism in the results due to the heterogeneity of the cluster, for simulations with nodes 1 to 6, slave programs have always been run on the "fast" (2.7 GHz) machines, and for simulations with 7 to 10 nodes additional "slow" (1.8 GHz) machines have been used to complete the required number of nodes. The master program has always been run on the master node of the cluster (1.8 GHz).

Three different system sizes and different number of partitions –one per worker processor– have been considered for the experiments. Figure 10 shows results, and a significant runtime decrease can be noticed when the number of processors increase. The only exception is the change from 6 to 7 processors where an increase is registered due to the assigning strategy that has been used: only "fast" nodes are assigned to jobs with 6 or less processors and for jobs with more than 6 processors some "slow" processors have to be used.

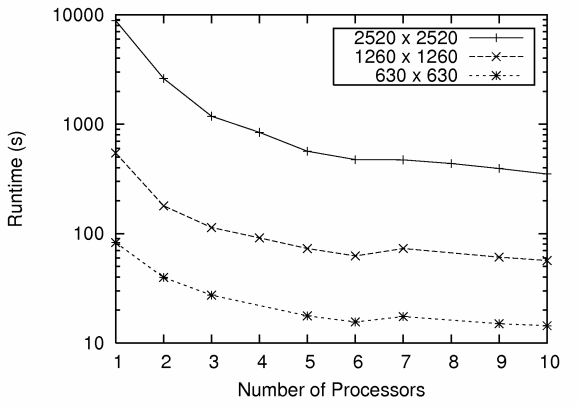

Fig. 10 Runtime of the experiments, using a logarithmic scale, for different number of partitions of the whole CA, each running on a different processor. Measurements for three different system sizes are shown.

In order to measure the performance of a parallel application, *speedup* (S_p) can be employed as defined by Foster in [35]. This value establishes a comparison between a parallel algorithm and its sequential counterpart. It can be defined as the ratio of the runtime of the sequential version of the program running on 1 processor of the parallel computer (T_1) to the runtime of the parallel version running on m processors of the same computer (T_m):

$$S_p(m) = \frac{T_1}{T_m} \tag{4}$$

Fig. 11 shows the speedup obtained for the parallel implementation of our CA model for the three different system sizes, compared to the *linear speedup* –line $y = x$. For the smallest system size, a very good performance has been obtained. For the other two system sizes, still better performance figures are obtained, in fact *super-linear speedup* (speedup higher than linear). The main reason is the finite memory space available for only one processor and therefore the necessity of using the swap memory. Because of this circumstance, the calculation of very large system sizes –as for example a detailed 3D simulation– may not be affordable on a single PC (for the prohibitively large runtime needed due to the use of swap memory) but feasible on a cluster, in which the system is partitioned so each individual node

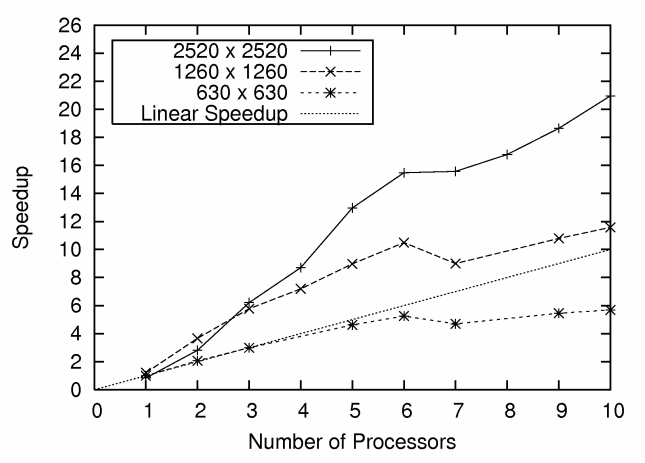

Fig. 11 Speedup obtained for the parallel implementation with respect to the sequential program for different number of processors and for three different system sizes. For comparison, the ideally optimal linear speedup has been shown. A very good performance is obtained for a moderate system size (630×630 cells) and a super-linear speedup for larger system sizes.

needs less memory and does not have to use swap memory (interested readers can also check [9]).

In order to analyze the scalability of the combination parallel application-parallel computer, the running times obtained for the same experiment when increasing the system size (which in our problem is represented by the total number of cells of the CA) and the number of processors by the same factor have been compared. The results are shown in Figure 12. The same experiment as for Fig. 4(b) has been used, but involving the computation of 10000 time steps. For an ideally scalable application, the same running time should be obtained [36]. Our parallel application shows only a small excess (from 2 % to 5 %) of runtime compared to the optimal value. Therefore its scalability is good on a small computer cluster.

5.4 Load Balancing with Dynamite

Previous sections have shown the interest of deploying a parallel version of the CA based laser model. We consider now the problem of load balancing: how to obtain the best results when running experiments on a non-dedicated platform, where different tasks dynamically arrive to run simultaneously with our parallel application. The problem is thus to decide when and where different tasks must be migrated for improving speedup.

In order make this study we have included artificial loads that simulate a normal non-dedicated cluster use. All the experiments shown below have thus been run

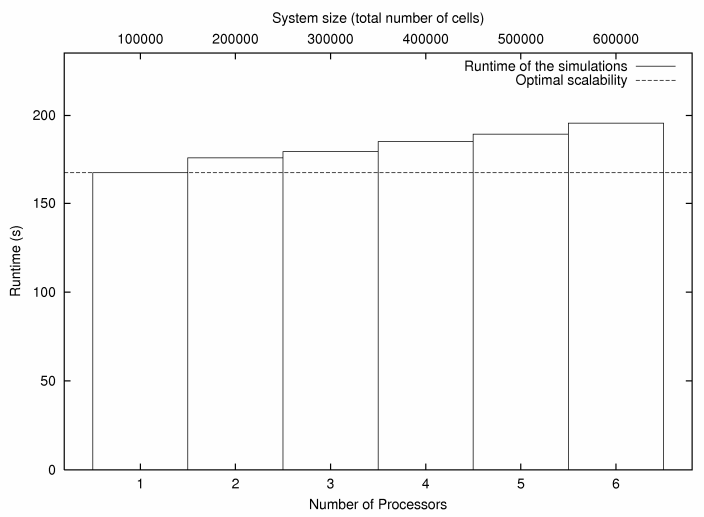

Fig. 12 Scalability analysis of the combination parallel application-parallel computer. The runtime for the same experiment but increasing the system size and the number of processors by the same factor is shown. The optimal ideal value would be the same runtime for all cases (horizontal line). The results show only a small excess (from 2 % to 5 %) with respect to this optimal value.

under controlled conditions on the cluster. We have considered the computation of the time evolution of the system during $10,000$ time steps for a single value of the system parameters: $\lambda = 0.0125$, $\tau_c = 10$, $\tau_a = 180$.

A sequential program with a loop statement including a single assign instruction involving double precision numbers was employed as the external load, similarly as described in [34]. This C program was compiled regularly with no application of optimization techniques that would allow to improve the runtime of the program. The artificial load was intended to simulate the normal use of a non-dedicated high performance computing cluster for different users. Normally, to achieve the best performance possible, a cluster user would not run more than one process of her application on any cluster node. For that reason, only one artificial load process was executed on each cluster node.

The experiments used 6 worker nodes plus the master one for the parallel CA application, while 10 nodes were available on the cluster. The external load was systematically assigned to a number of cluster nodes, and time was then measured. The idea was to study the effect of different levels of loads for both the regular PVM version of the algorithm, and also the one employing Dynamite, which includes the load balancing system.

Immediately after starting the CA application, the artificial load task was initiated on a number of nodes, which range from 0 to 5 nodes and were always nodes to which one of the slave CA applications had been initially allocated also. The artificial load tasks kept on running for a time longer than the total execution time of the CA application.

Table 2 Execution time and improvement due to load balancing when the application is run with and without load balancing and running artificial external load on a different number of cluster nodes. Normal PVM was used for configurations without load balancing and the Dynamite system for configurations with load balancing.

Configuration	Execution time (s)	Improvement
No load balancing with artificial load	1895.08	-
Load balancing with load on 1 node	384.59	80 %
Load balancing with load on 2 nodes	564.76	70 %
Load balancing with load on 3 nodes	611.12	68 %
Load balancing with load on 4 nodes	1595.75	16 %
Load balancing with load on 5 nodes	1833.82	3 %
No load, with and without load balancing	233.43	-

5.4.1 Results and Discussion

Table 2 presents results obtained for the experiments described above, including execution time and improvement obtained when the load balancing technique is employed.

The first row shows the execution time obtained when no load-balancing technique is employed –standard PVM– while external loads are applied to any number of nodes, from 1 to 5. Regardless of the number of nodes undergoing external loads, the execution time is always the same. The reason is that the CA laser model operates in a lock-step mode, and the slowest running task limits the global performance obtained.

The following rows show the execution time when Dynamite is employed – instead of regular PVM, so that the load balancing mechanism is enabled. External artificial loads are applied again to nodes ranging from 1 to 5. Additionally, each row offers information about the improvement obtained when compared to previous execution –first row, when no load-balancing technique was employed.

Finally, the execution time obtained when running the application without any artificial load, which is the same with and without load balancing, has been shown as a reference in the last row.

Fig. 13 represents the number of executed time steps from the application versus time, which offers an idea of the progression of the application. We can notice an interesting improvement in the performance obtained when the parallel application is run on 6 worker nodes plus the master one, while a total number of 10 nodes are available on the cluster: when external load is run on up to 3 nodes, and given that idle nodes are available, the load balancing mechanism takes a good advantage, migrating required tasks, therefore reducing the execution time by a factor or 3. If more than 3 nodes receive external load, the improvement obtained is lower, but we still obtain a smaller execution time when compared to the first row of the table. The execution progress initially follows the same straight line as for no dynamic load balancing (i.e. for standard PVM), until the load balancing system identifies the

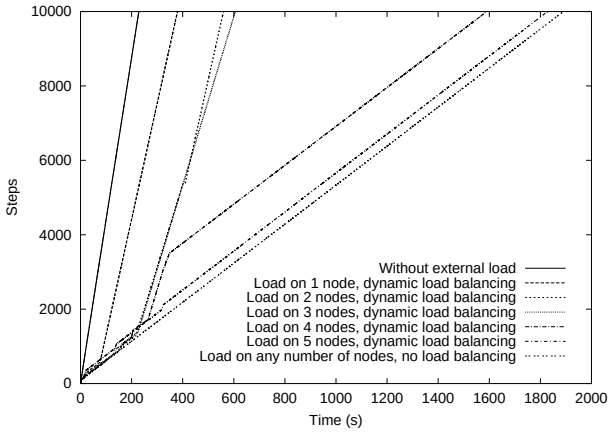

Fig. 13 Execution progress of the CA laser model application for different levels of artificial external load on the system. The system size was 840×840 cells. The number of cluster nodes used on the execution is 6.

situation and performs the migration of some of the tasks of the system to balance the load.

After that, when external loads are run on a small number of nodes, a significant improvement is found in the execution progress, following again a new straight line close to that one of the standard PVM. When considering external loads on a higher number of nodes, sometimes the benefits obtained after migrations produced by the load balancing mechanism are very low.

We have also found that occasionally, after an advantageous migration of tasks, the dynamic load balancing system incorrectly migrates tasks to let the system load unbalanced and obtain a sub-optimal execution progress.

It is also of interest to point out that the dynamic load balancing system incurs in practically no overhead on the execution time of the application, as its execution progress is virtually identical for PVM and Dynamite when there is no external load applied: the same line in Fig. 13 (labeled as ”Without external load”) applies to both cases.

We have also studied the effect of the system size on the global performance obtained. To this end, we have run simulations for three different system sizes and the execution progress has been compared.

The results of the experiments performed are shown in Fig. 14. Given that large CA sizes might require the use of swap memory of the operating systems, which would greatly decrease the whole performance of the system, keeping us from correctly analyzing runtime, we have employed relatively small CA sizes (interested readers can also see [11]). The figure shows that the use of a load balancing strategy results in a good performance improvement for all system sizes within the studied ranges.

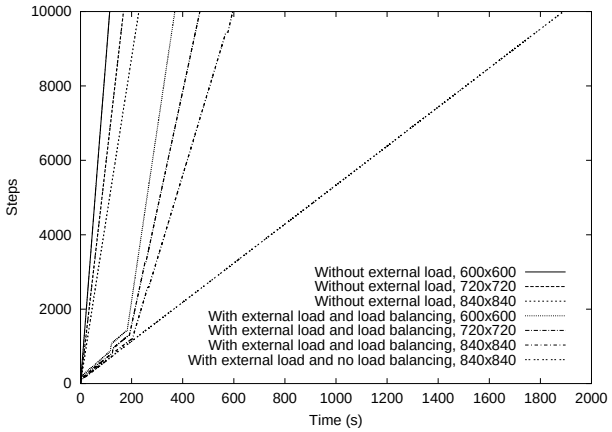

Fig. 14 Execution progress of the CA laser model application for different system sizes. The number of cluster nodes used on the execution is 6 and artificial external load has been run on 3 nodes.

Finally, we have studied the frequency and regularity of activation of the scheduling mechanism, so that a given experiment has been performed a number of times under exactly the same initial conditions.

Fig. 15 shows four different runs of the same experiment using Dynamite. We can notice some cases in which the load balancing system lets the load unbalanced and the execution time is not optimal. Although this behavior was also present in the experiments reported in previous figures for a 10% - 20% of the executions, these cases were not taken into account for the results presented.

We can thus conclude that migrations are not performed by the load balancing system in a very regular and deterministic way, and although results obtained are globally of interest , we clearly see that the scheduler component of the Dynamite load balancing system could be improved.

6 Future Work: Virtualization and load balancing

In the previous sections, we have reviewed the application of cellular automata to model laser dynamics and we have shown that with this approach it is possible to develop high performance simulations that run efficiently on computer clusters. We have also shown that these simulations can successfully be run on heterogeneous non-dedicated clusters, using an adequate load balancing mechanism, with a good performance. Finally, in this section we will present some ideas about future work in this subject, regarding the joint use of a dynamic load balancing tool like Dynamite and virtualization technology, to have a self-adapting cluster computing environ-

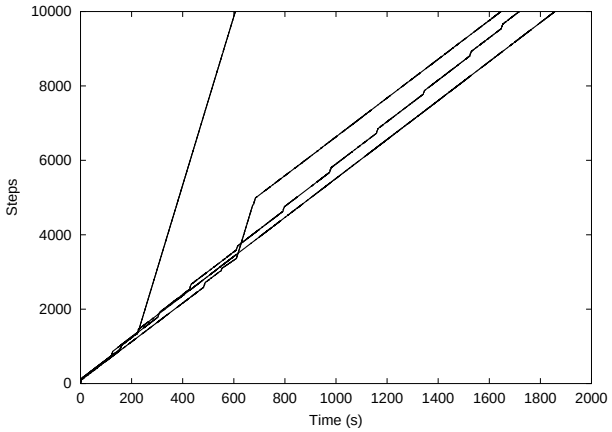

Fig. 15 Execution progress of 4 different runs of the application with Dynamite carried out under the same conditions. System size: 840×840 cells.

ment capable of deploying additional cluster nodes on demand, in the course of a computation.

Nowadays, the virtualization technology is gaining more adepts quickly. The benefits of using it as a solution for deploying and administering different services like HTTP, FTP, etc. within the virtual machines, have achieved a great success. Big processor companies like Intel or AMD have their own and specific solutions to provide a good performance within the virtualization paradigm.

Virtualization is a technology that allows to multiplex the physical hardware to take advantage of its computer resources [37, 38, 39]. This technology creates a "virtual" machine (VM) where it is possible to load and run an operating system and its associated applications, for example a cluster node.

The VM is handled by a software called the *virtual machine monitor* (VMM) or hypervisor. The VMM is in charge of virtualizing the underlying hardware and assures that a problem or a bug within the VM will affect only to the VM and not to the real hardware and OS. Therefore, security and isolation are two of the main benefits of using VMs.

The main features of virtualization are the following:

- **Resource isolation**. Virtualization isolates each VM inside the host machine. This is very useful from the standpoint of the researcher because a failure inside a VM will affect only that VM and not the real machine.
- **Guest OS instantiation**. This feature permits to create an OS image that can be loaded into any machine that is compatible with the VMM employed for creating it.
- **Snapshots** or state serialization (also known as checkpointing [40]). With virtualization it is possible to freeze the execution of a whole OS and restart it exactly where it was stopped.

There are two different approaches to provide virtualization in x86 platforms. One of them is the *native virtualization*. The native virtualization implements this technology by providing an exact copy of the underlying hardware for the VM, in terms of functionality. This approach is two-folded: any OS is supported without modifications, but the performance gets affected due to x86 was not designed to be virtualized [41]. For this reason, any problematic instruction has to be captured by the VMM to assure the right operation of the virtual machine.

The other approach is the *paravirtualization*. This technique implements virtualization by providing a virtual hardware that is similar, instead of identical, to the underlying hardware in order to circumvent the previous described x86 problem. In order to use this technique, the guest OS has to be adapted to support the paravirtualization while applications can be run without any modifications (the binary interface is not modified).

Both techniques have different products and software solutions, for instance: VMware [37, 42], Xen [38] or VirtualBox [39]. Additionally, the processor manufacturers are also interested in virtualization and they are providing specific solutions for virtualization in their microchips (Intel VT-x[1] technology and AMD-V Pacifica [2]). Thanks to this new technology, the paravirtualization and native virtualization solutions can improve its performance and features.

To sum up, the virtualization is a promising technology that can improve the deployment and maintenance of clusters, due to its main features: resource isolation, guest OS instantiation and snapshots. On the other hand, take into account research groups or institutions that have clusters or additionally servers dedicated to other services like HTTP, FTP, etc. Despite of the load of those machines, it could be interesting to take advantage of the computing resources that these machines can add to our cluster by means of VMs. The goal is to install virtual cluster nodes on non-dedicated cluster nodes to obtain computing resources when requested and available.

To the best knowledge of the authors, several attempts have been done on integrating virtualization and clusters. J.S. Chase et. al. [43] present new mechanisms for dynamic resource management in a cluster manager called Cluster-On-Demand. I. Foster et. al. [44] propose to give custom client clusters to circumvent the hardware and software heterogeneity of clusters. Finally, W. Emeneker et. al. [45] propose to use virtualization in clusters for job forwarding and spanning. However, none of the above cited articles use Dynamite, the load balancing tool employed on this work. Therefore, what we are considering in the context of the problem presented in this chapter is the deployment of additional cluster nodes on-demand by means of virtualization. The goal is to improve Dynamite (see Section 5.4) adding a new virtualization feature that can request more computing power by launching virtual nodes on other machines, for example other servers from the institution. The application will determine when it is necessary more computer power and request it by launching those virtual cluster nodes.

[1] `http://www.intel.com/technology/virtualization/`

[2] `http://www.amd.com/us-en/0,,3715\_15781,00.html?redir=wsv08`

This improvement to Dynamite will benefit the whole cluster as virtualization gives identical cluster computing nodes as real ones. It is obvious that using a virtualization technology gives some overhead [38, 37], but thanks to new microprocessors from Intel and AMD and the improvements on the technology, virtual nodes performance is more or less equal to real hardware [38, 37].

Additionally, the use of virtualization technology will allow to checkpoint and migrate any running virtual cluster node without having to implement any special library for checkpointing and migration at level process like Overeinder et. al. proposed on his work [30]. Virtualization simplifies this problem by checkpointing the whole node and migrating it without losing any kind of connectivity or data (see [38]). Furthermore, the possibility of serializing and migrating a VM (snapshots) opens new opportunities to load balancing and reliability.

Other benefit of using VMs is the possibility of running cluster nodes on different OSs and architectures because virtualization abstracts the underlying hardware. Thus, it will be possible to run GNU/Linux cluster nodes on Microsoft Windows machines without any kind of problem. Furthermore, thanks to virtualization it will be possible to harness all the computers from an institution or research laboratory independently of its OS platform, providing more computing power to the cluster when necessary. Bear in mind that Dynamite will launch virtual machines only when more computing power is required.

In conclusion, thanks to this new approach it will be possible to have a more powerful and flexible cluster that auto-adapts itself to the CPU load launching or stopping virtual nodes on-demand.

7 Conclusions

This chapter has presented the modeling of a well-known complex system, the laser, by means of a parallel version of a bioinspired algorithm, the cellular automaton.

By means of a series of experiments, we have considered key factors of the parallel algorithm when running on clusters of workstation. We have used a cluster computing environment for being better suited in general than a grid computing platform to run a parallel CA due to its lower latency on the communications.

Firstly we have shown the feasibility of CA for modeling the laser. Secondly, we have studied the performance obtained by a parallel version of the model, and finally we have considered the execution of the algorithm on a non-dedicated cluster, when external loads dynamically arrive while the CA tasks are being simultaneously run.

We have thus shown that the parallel version of the algorithm –following a master-worker model using the message passing mechanism- can offer good scalability when running on a cluster, which is of interest for running large versions of the CA model for modeling realistic laser systems.

We have then moved to a more realistic scenery by considering the presence of external loads on the cluster system. We have evaluated the performance of the application including artificial external loads to simulate the effect of other tasks

running simultaneously on the cluster. In this case, a dynamic load balancing strategy has been used, with two main differences with respect to most previous parallel CA implementations: load can be migrated to new nodes initially not belonging to the pool and the load balancing functionality is uncoupled from the CA algorithm. For this purpose, we have run the parallel application on top of a dynamic load balancing software tool –Dynamite–. This modular approach has the advantage that changes can be introduced to the CA algorithm or to the dynamic load balancing strategy without perturbing each other.

In spite that for this kind of application –a synchronous cellular automaton– all the computing nodes must have finished an iteration before the next one can be initiated, the results have been very satisfactory. The performance of the parallel application is improved by the load balancing strategy from 60% to 80% when there are some idle nodes on the cluster to which some load can be migrated. Still when there were no such idle nodes, the execution time was always shorter than without the use of load balancing.

In conclusion, we have reviewed the application of a parallel cellular automata model to simulate laser dynamics and we have also presented evidence of the feasibility of running large parallel simulations using this approach to simulate realistic laser devices on heterogeneous non-dedicated clusters if an adequate dynamic load balancing strategy is used.

References

1. A. Einstein. Zur quantenmechanik der strahlung. Physikalische Zeitschrift, 18:121–128. 1917.
2. A.E. Siegman. Lasers. University Science Books, Mill Valley, CA. 1986.
3. J.L. Guisado, F. Jiménez-Morales, J.M. Guerra. Cellular automaton model for the simulation of laser dynamics. Physical Review E,67(6): 066708. 2003.
4. A. Ilachinski. Cellular automata. A discrete Universe. World Scientific, Singapore, 2001.
5. P. M. A. Sloot and A. G. Hoekstra. Modeling Dynamic Systems with Cellular Automata, chapter 21, pages 21–1+6. Chapman & Hall/CRC, 2007. ISBN 1-58488-565-3.
6. Bastien Chopard and Michel Droz. Cellular Automata Modeling of Physical Systems. Cambridge University Press, Cambridge, 1998.
7. J.L. Guisado, F. Jiménez-Morales, J.M. Guerra. Application of shannon's entropy to classify emergent behaviors in a simulation of laser dynamics. Mathematical and Computer Modelling,42: 847–854. 2005.
8. J.L. Guisado, F. Jiménez-Morales, J.M. Guerra. Computational simulation of laser dynamics as a cooperative phenomenon. Physica Scripta, T118: 148–152. 2005.
9. J.L. Guisado, F.Jiménez-Morales, and F.Fernández de Vega. Cellular automata and cluster computing: An application to the simulation of laser dynamics. Advances in Complex Systems, 10(Suppl. No. 1):167–190, 2007.
10. J.L. Guisado, F.Fernández de Vega, and K.Jiménez-Morales, F.and Iskra. Parallel implementation of a cellular automaton model for the simulation of laser dynamics. Lecture Notes in Computer Science, 3993:281–288, 2006.
11. J. L. Guisado, F. Fernández de Vega, and K. Iskra. Performance analysis of a parallel discrete model for the simulation of laser dynamics. In 2006 International Conference on Parallel Processing, Workshops, pages 93–99. IEEE Computer Society, 2006.

12. J.L. Guisado, F. Fernández de Vega, F. Jiménez-Morales, K.A. Iskra, P.M.A. Sloot. Using cellular automata for parallel simulation of laser dynamics with dynamic load balancing. International Journal of High Performance Systems Architecture, 1(4): 251–259. 2009.
13. D.Talia. Cellular processing tools for high-performance simulation. IEEE Computer, 33(9):44–52, 2000.
14. M. Resnick. Turtles, Termites, and Traffic Jams. MIT Press, Cambridge, Mass, USA. 1994.
15. M. Cannataro, S. Di Gregorio, R. Rongo, W. Spataro, G. Spezzano, D. Talia. A parallel cellular automata environment on multicomputers for computational science. Parallel Computing,21(5): 803–823. 1995.
16. G. Spezzano, D. Talia, S. Di Gregorio, R. Rongo, W. Spataro. A parallel cellular tool for interactive modeling and simulation. IEEE Computational Science & Engineering,3(3): 33–43. 1996.
17. D. Hutchinson, L. Kattner, M. Lanthier, A. Maheshwari, D. Nussbaum, D. Roytenberg, J.R. Sack. Parallel neighbourhood modeling: research summary. Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures: 204–207. 1996.
18. L. Carotenuto, F. Mele, M. Furnari, R. Napolitano. PECANS: A parallel environment for cellular automata modeling. Complex Systems,10(1): 23–42. 1996.
19. B. Zeigler, Y. Moon, D. Kim, G. Ball. The DEVS environment for high-performance modeling and simulation. IEEE Computational Science & Engineering,4(3): 61–71. 1997.
20. A. Schoneveld and J.F. de Ronde. P-CAM: a framework for parallel complex systems simulations. Future Generation Computer Systems, 16(2):217–234, 1999.
21. T. Toffoli and N. Margolus. Cellular automata machines: a new environment for modeling. MIT Press, Cambridge, MA. 1987.
22. P.M.A. Sloot, J.A. Kaandorp, A.G. Hoekstra, and B.J. Overeinder. Distributed simulation with cellular automata: architecture and applications. Lecture Notes in Computer Science, 1725:203–248, 1999.
23. D.D'Ambrosio and W.Spataro. Parallel evolutionary modeling of geological processes. Parallel Computing, 33(3):186–212, April 2007.
24. M. Mazzariol, B. Gennart, and R. Hersch. Dynamic load balancing of parallel cellular automata. In Proc. SPIE Conference on Parallel and Distributed Methods for Image Processing IV, volume 4118, page 2129, San Diego, July 2000. SPIE.
25. G. A. Kohring. Dynamic load balancing for parallelized particle simulations on MIMD computers. Parallel Computing, 21:683–693, 1995.
26. A. Cortés, M. Planas, J. L. Millán, A. Ripoll, M. A. Senar, and E. Luque. Applying load balancing in data parallel applications using DASUD. Lecture Notes in Computer Science, 2840:237–241, 2003. Euro PVM/MPI 2003.
27. J. C. Fabero, I. Martin, A. Bautista, and S. Molina. Dynamic load balancing in a heterogeneous environment under PVM. In 4th Euromicro Workshop on Parallel and Distributed Processing (PDP '96), pages 414–419. IEEE Computer Society, 1996.
28. J. R. Weimar. Cellular automata for reaction-diffusion systems. Parallel Computing, 23(11):1699–1715, 1997.
29. G. Dick van Albada, J. Clinckmaillie, A. H. L. Emmen, Jörn Gehring, O. Heinz, Frank van der Linden, Benno J. Overeinder, Alexander Reinefeld, and Peter M. A. Sloot. Dynamite - blasting obstacles to parallel cluster computing. Lecture Notes In Computer Science, 1593:300–310, 1999. HPCN Europe '99: Proceedings of the 7th International Conference on High-Performance Computing and Networking.
30. B. J. Overeinder, P. M. A. Sloot, R. N. Heederik, and L. O. Hertzberger. A dynamic load balancing system for parallel cluster computing. Future Generation Computer Systems, 12(1):101–115, 1996.
31. Kamil Iskra, Zeger W. Hendrikse, G. Dick van Albada, Benno J. Overeinder, Peter M. A. Sloot, and Jörn Gehring. Experiments with migration of message-passing tasks. Lecture Notes in Computer Science, 1971:203–213, 2000. GRID '00: Proceedings of the First IEEE/ACM International Workshop on Grid Computing.

32. Kamil Iskra, Zeger W. Hendrikse, G. Dick van Albada, Benno J. Overeinder, and Peter M. A. Sloot. Dynamic migration of PVM tasks. In ASCI 2000, Proceedings of the sixth annual conference of the Advanced School for Computing and Imaging, pages 206–212, June 2000.
33. G. Folino and G. Spezzano. An autonomic tool for building self-organizing grid-enabled applications. Future Generation Computer Systems, 23(5):671–679, 2007.
34. S. S. Vadhiyar and J. J. Dongarra. Self adaptivity in grid computing. Concurrency Computation Practice and Experience, 17(2-4):235–257, 2005.
35. I. Foster. Designing and building parallel programs. Addison-Wesley, Reading, MA. 1995.
36. Dongarra, J., Foster, I., Fox, G. C., Gropp, W., Kennedy, K., Torczon, L. and White, A. (eds.). Sourcebook of parallel computing (Morgan Kaufmann, San Francisco, 2003).
37. J. Sugerman, G. Venkitachalam, B. Lim. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor.
38. P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield. Xen and the art of virtualization, Proceedings of the nineteenth ACM symposium on Operating systems principles (2003) 164–177.
39. J. Watson. Virtualbox: bits and bytes masquerading as machines, Linux J. 2008 (166),1 (2008).
40. E. Elnozahy, L. Alvisi, Y. Wang, D. Johnson. A survey of rollback-recovery protocols in message-passing systems, ACM Computing Surveys (CSUR) 34 (3) (2002) 375–408.
41. J. Robin, C. Irvine, N. P. S. M. C. D. O. C. SCIENCE. Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor, Defense Technical Information Center, 2000.
42. J. Nieh, O. C. Leonard. Examining VMware, j-DDJ 25 (8) (2000) 70, 72–74, 76.
43. J.S. Chase, D.E. Irwin,L.E. Grit, J.D. Moore, S.E. Sprenkle. Dynamic Virtual Clusters in a Grid Site Manager, HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (2003) 90.
44. I. Foster,T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, X. Zhang. Virtual Clusters for Grid Communities, CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (2006) 513–520.
45. W. Emeneker, D. Stanzione. Dynamic Virtual Clustering, 2007 IEEE International Conference on Cluster Computing, (2007) 84–90.