Parallel Implementation of a Cellular Automaton Model for the Simulation of Laser Dynamics

J.L. Guisado[1], F. Fernández de Vega[1], F. Jiménez-Morales[2], and K.A. Iskra[3]

[1] Centro Universitario de Mérida, Universidad de Extremadura,
Sta. Teresa Jornet, 38. 06800 Mérida (Badajoz), Spain
http://cum.unex.es/profes/profes/jlguisado
[2] Departamento de Física de la Materia Condensada, Universidad de Sevilla,
P.O. Box 1065, 41080 Sevilla, Spain
[3] Section Computational Science,
Faculty of Science, Universiteit van Amsterdam,
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands

Abstract. A parallel implementation for distributed-memory MIMD systems of a 2D discrete model of laser dynamics based on cellular automata is presented. The model has been implemented on a PC cluster using a message passing library. A good performance has been obtained, allowing us to run realistic simulations of laser systems in clusters of workstations, which could not be afforded on an individual machine due to the extensive runtime and memory size needed.[1]

1 Introduction

In the last two decades, computational simulations based on cellular automata (CA) have been extensively used in many fields of science and technology [1]. More recently, many parallel implementations of CA models have been presented [2]. One of the reasons is that CA are intrinsic parallel systems very suitable to be easily implemented in parallel computers to carry out high performance simulations. Another reason is that in the same period parallel computer architectures have experienced a huge development and a "democratization" due to the affordability of clusters of workstations with a very good price/performance ratio. As a consequence, parallel CA simulations have been successfully applied in many fields, see for example [3, 4, 5, 6]. In addition, different software tools for the programming of CA in parallel computers (for example [7, 8]) have been introduced.

In this work, we present a parallel implementation in two dimensions of a discrete model of laser dynamics based on CA, introduced in references [9, 10, 11]. This implementation will allow us to run large size 2D simulations of the model on clusters of workstations. In addition, the 2D implementation will be useful to test the feasibility of a parallel 3D version of the model, needed to make realistic

[1] This work was partly supported by the project OPLINK (TIN2005-08818-C04-03) of Ministerio de Educacin y Ciencia (Spain).

V.N. Alexandrov et al. (Eds.): ICCS 2006, Part III, LNCS 3993, pp. 281–288, 2006.

simulations of specific laser systems, which would necessarily require a parallel implementation due to its extensive runtime and memory requirements.

The rest of the paper is organized as follows. In Section 2 the discrete model for the simulation of laser dynamics is summarized. The parallel implementation of the model is described in Section 3. In Section 4 the performance of our implementation is analyzed. Finally, the conclusions of this study are explained in Section 5.

2 Cellular Automaton Model

A laser system is modeled by a cellular automaton [9, 10] defined on a two-dimensional square lattice of $N_c = L \times L$ cells with periodic boundary conditions. Two variables $a_i(t)$ and $c_i(t)$ are associated with each node of the CA. $a_i(t)$ represents the state of the electron in node i at time t: if $a_i(t) = 0$ the electron is in the laser ground state and if $a_i(t) = 1$ it is in the upper laser state. $c_i(t) \in \{0, 1, 2, ..., M\}$ represents the number of photons in node i at time t. A large enough upper value of M is taken to avoid saturation of the system. The state variables values, which represent "bunches" of real photons and electrons, are obviously smaller than the real number of photons and electrons in the system and connected to them by a normalization constant. The *Moore neighborhood* is considered. The transition rules, which represent the different physical processes in a laser system at the microscopic level, are:

- Rule 1. Pumping: If $a_i(t) = 0$ then $a_i(t + 1) = 1$ with a probability λ.
- Rule 2. Stimulated emission: If $a_i(t) = 1$ and the sum of the values of the laser photons states in the nine neighbor cells is greater than a certain threshold (1 in our model), then $c_i(t + 1) = c_i(t) + 1$ and $a_i(t + 1) = 0$.
- Rule 3. Photon decay: A finite life time τ_c is assigned to each photon when it is created. The photon will be destroyed τ_c time steps after it was created.
- Rule 4. Electron decay: A finite life time τ_a is assigned to each electron that is promoted from the ground level to the upper laser level. That electron will decay to the ground level again τ_a time steps after it was promoted, if it has not yet decayed by stimulated emission.

Spontaneous emission as well as thermal contributions are simulated by a continuous noise of random photons introduced at every time step in the laser mode, by making $c_i(t+1) = c_i(t)+1$ for a small number N_n of cells ($< 0.01\%$ of total) with randomly chosen positions. As in real lasers, these random photons are responsible of the initial start-up of the laser action.

3 The Parallel Implementation

A very large automaton must be used to obtain results that quantitatively reproduce the behaviour of lasers (macroscopic systems) because a fine-grained model is needed, or in general to implement any 3D model. The runtime for a

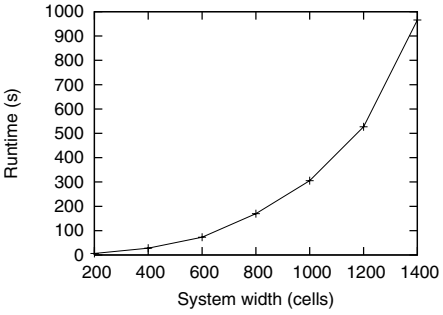

Fig. 1. Runtime of an experiment consisting in running a sequential implementation of the 2D laser model for 1000 time steps, using the same values of the parameters, for different system sizes

typical experiment grows quickly with the automaton size, as shown in Fig. 1. Therefore a parallel implementation is mandatory.

Parallelization of the model has been performed for distributed-memory MIMD (multiple-instruction multiple-data) systems using the message passing paradigm. The Parallel Virtual Machine (PVM) library has been used, because we were interested in a further study of our model using dynamic load balancing mechanisms specifically developed for this library. Nevertheless, it would be straightforward to port this implementation to other message passing libraries such as Message Passing Interface (MPI). Parallelization has been carried out following the *master-slave* programming model and the *data decomposition* or *partitioning* methodology for workload allocation: identical tasks operate on different portions of the data. A "*master program*" divides the CA grid in p partitions of equal size and sends each to a "*slave program*" running on a different processor. The particular tasks performed by the master and slave programs are:

- Master program:
  1. Input data from external file (system size, number of partitions, parameter values, number of time steps) and initialization.
  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 for each time step.
  6. Termination of slave programs.
  7. Calculations with the complete results.
  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.

A diagram of this procedure is shown in Fig. 2 (a), where each box in bold type represents a different processor and the bold arrows represent the communications between the processes running on different processors.

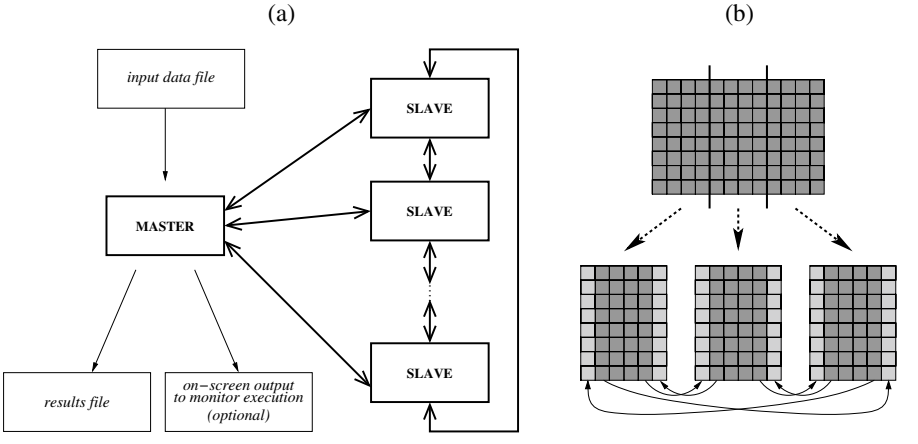

Fig. 2. (a): Block diagram of the parallel computing structure used. (b): The CA has been partitioned using a 1-dimensional domain decomposition: the automaton is vertically partitioned in stripes and each sub-domain is assigned to a different processor. Two additional columns of *ghost cells* are added at both sides of each partition to store the photon state $c_i(t)$ of neighboring cells belonging to different partitions.

In general, for d-dimensional CA 1-, 2-, ... or d-dimensional domain decompositions can be considered. Here, a 2D CA is used so that two possibilities arise: using a 1D domain decomposition, i.e. partitioning the CA in stripes, or using a 2D (checkerboard) domain decomposition. We have used a 1D domain decomposition because it makes the communication structure simpler and minimizes the number of send/receive calls (essential for an implementation using message passing). Furthermore, it is more favorable in runtime for a small to moderate number of nodes [12], despite the amount of data to be communicated is larger. As shown in Fig. 2 (b), the CA is vertically partitioned in stripes and each sub-domain is assigned to a different processor. For each sub-domain, two additional columns of ghost cells have been included at the left and right sides, used to store the photon state $c_i(t)$ of neighboring cells belonging to different sub-domains.

Each slave program computes the time evolution on its assigned partition of the automaton by applying the CA evolution rules. In each time iteration, the application of the evolution rules involves the following procedures:

1. Stimulated emission.
2. Refresh values of photon state $c_i(t)$.
3. Photon and electron decay and electron pumping.
4. Noise photons.

Procedures 1 to 3 are three successive loops through all of the cells in the partition. In the first one, stimulated emission is computed storing new values of the photon state $c_i(t)$ in a temporal array; in the second one, the values of the photon state $c_i(t)$ of all the cells in the partition are updated using the values stored in the temporal array; in the third one, photon and electron decay and electron pumping are computed. In procedure 4, N_n/p noise photons are introduced in randomly chosen cells inside the partition, where N_n is the total number of noise photons introduced in the system at every time step and p is the number of partitions in the system.

For CA in general, after each time iteration, the state of the CA cells in the boundaries of each slave partition must be communicated to the slave programs dealing with the neighboring partitions, because this state will be needed to compute the CA evolution rules there. For this particular model, the only state value from neighboring cells needed to compute the CA evolution rules for a cell is the photon state $c_i(t)$. Therefore, only this state is communicated to the neighboring partitions. This communication is carried out directly between the slave programs. In addition, in the last part of each time iteration, each slave program computes the total number of electrons $a_i(t)$ and laser photons $c_i(t)$ in its CA partition and sends this information to the master, which can record it and make some calculations with it, such as computing the Shannon's entropy as described in [9].

4 Performance Analysis

In order to test the performance of the parallel implementation, simulations have been carried out on a Linux PC cluster of ten nodes with Intel Pentium-4 processor, for different sizes of the cellular automaton grid (2520×2520, 1260×1260 and 630×630 cells). The cluster is heterogeneous: six nodes have a clock frequency of 2.7 GHz and the other four nodes of 1.8 GHz. A particular configuration has been chosen for the experiments to avoid indeterminism in the results. For simulations with 1 to 6 nodes, the slave programs have been run on the "fast" (2.7 GHz) machines. For simulations with 7 to 10 nodes, additional "slow" (1.8 GHz) machines have been used to achieve the required number of nodes. The master program was always run on the master node of the cluster (which runs at 1.8 GHz).

The results of the simulations carried out with a sequential implementation of the 2D CA laser model (previously presented in references [9, 10]) are reproduced with the present parallel implementation. An example is shown in Fig. 3 which shows the time evolution for 1000 iterations (time steps) of the population inversion and the total number of laser photons for two different system

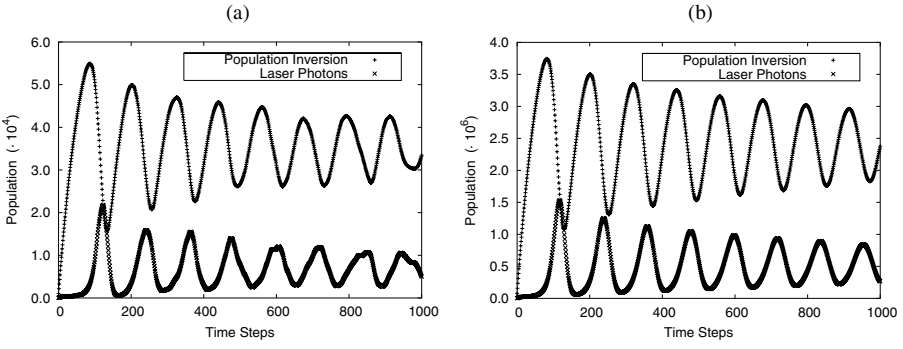

Fig. 3. Evolution of the whole system for two different system sizes. (a) (left): 300 × 300 cells, typical result of previous sequential implementations of the model. (b) (right): 2520 × 2520 cells, using the parallel implementation with a much larger system size.

sizes: 300 × 300 cells (a), typical system size for previous sequential implementations; and 2520 × 2520 cells (b). In both cases the values of the parameters are: $\lambda = 0.0125$, $\tau_c = 10$, $\tau_a = 180$. The ratio of noise photons (introduced in every time step) to total number of cells in the system has been maintained constant (0.03% of the cells) for the experiments with different system sizes. The typical laser behaviour known as *relaxation oscillations* or *laser spiking* (correlated large amplitude damping oscillations) [9] is reproduced. The differences between both results are that the populations are scaled and that the shape of the oscillations is smoother in (b) due to having used a much larger system size, thus reproducing more accurately the relaxation oscillations obtained in real lasers or from the integration of macroscopic differential equations.

The performance of the parallel implementation has been measured by running the same experiment (corresponding to the results shown in Fig. 3) using different system sizes and for different number of partitions of the whole CA (each one being handled by the slave program on a different node). The wall clock time of the experiments is shown in Fig. 4 (a), using a logarithmic scale. Runtimes get significantly shorter when the number of processors is increased, with the exception of the change from 6 to 7 processors. The reason is that "fast" machines have been used for a number of processors from 1 to 6, whereas "slow" machines are used to complete a number of processors higher than 6. As the CA operates in lock-step mode, the speed of the application is limited by the speed of the slowest running task and adding one "slow" machine results in a slower speed for the whole application.

The performance of the parallel implementation can be measured in terms of the *speedup*, ratio of the runtime of the sequential version of the program to the runtime of the parallel version. The speedup obtained for different number of partitions of the system is shown in Fig. 4 (b), in comparison with the line $y = x$, or *linear speedup*, which could be defined as the theoretical optimal speedup.

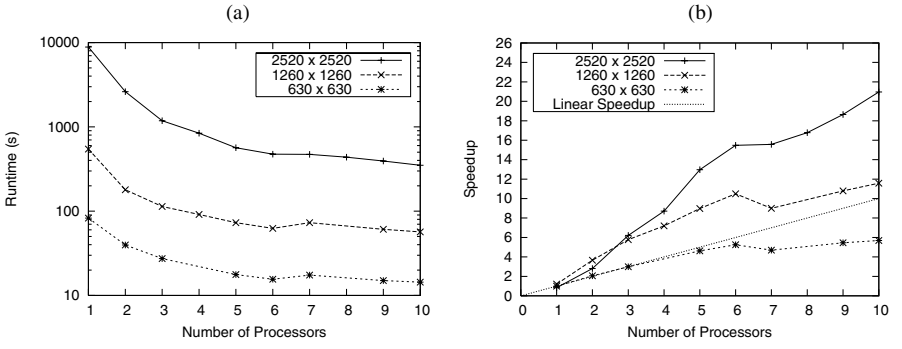

Fig. 4. (a): Runtime of the experiments, shown in a logarithmic scale, using three different system sizes, for different number of partitions of the whole CA. Each partition was run on a different processor. (b): Performance (in terms of speedup in respect to the sequential program) of the parallel implementation for varying number of processors and for three different system sizes.

A very good performance has been obtained. In fact, super-linear speedup is obtained, specially for a system size of 2520×2520 cells, and also for 1260×1260 cells. Swap memory is used for the sequential version of the program and not for the parallel version for more than one node. Therefore, a very high speedup, even higher than linear, can be obtained when comparing both magnitudes. This reason, together with similar finite memory cache effects, explains why super-linear speedup is produced. The use of swap memory for running a very large size simulation of the model (as necessary for 3D simulation) can make the calculation non-affordable on a single PC but feasible on a cluster, because the system is partitioned so that less memory is used in each node and no swap memory is needed.

In spite of the fact that, after every time step, the boundary data have to be exchanged and the intermediate results ($a_i(t)$ and $c_i(t)$) are transferred to the master program, a very good performance has been obtained. The reason is that, after the initial data are sent to all the slave processes and the computation begins, communication periods are much shorter than computation periods, as can be analyzed using a performance monitor. Therefore, the parallel application has a high computation-to-communication ratio (of the order of 10 for slave processes) and thus takes a good advantage of parallelization.

5 Conclusions and Future Prospects

A parallel implementation in 2D of a discrete model of laser dynamics using cellular automata has been presented. A CA model is an advantageous alternative to the standard description of laser dynamics, based on differential equations, when these have convergence problems —as for lasers ruled by stiff equations— or involve approximations not applicable —as for lasers with physical dimensions comparable to or even smaller than the wavelength of light, or

with an active medium of arbitrary geometry—. Another advantage of CA models is that its inherent parallel nature makes them very suitable to efficiently implement detailed simulations on parallel computers.

Parallelization has been carried out using the master-slave programming model and the data decomposition methodology for workload allocation. The parallel algorithm has been implemented for distributed-memory MIMD systems using the message passing paradigm. The implementation has been successfully run on a PC Cluster and its performance for different model sizes has been analyzed. Our implementation takes advantage of the parallelization and achieves a good speedup. This is a first step to run more realistic 3D models on clusters of workstations which could not be afforded on an individual machine due to the extensive runtime and memory size needed.

References

1. Chopard, B., Droz, M.: Cellular automata modeling of physical systems. Cambridge University Press (1998)
2. Bandini, S., Mauri, G., Serra, R.: Cellular automata: from a theoretical parallel computational model to its application to complex system. Parallel Computing **27**(5) (2001) 539–553
3. Sloot, P., Kaandorp, J., Hoekstra, A., Overeinder, B.: Distributed simulation with cellular automata: architecture and applications. Lecture Notes in Computer Science **1725** (1999) 203–248
4. Bandini, S., Magagnini, M.: Parallel processing simulation of dynamic properties of filled rubber compounds based on cellular automata. Parallel Computing **27**(5) (2001) 643–661
5. Dattilo, G., Spezzano, G.: Simulation of a cellular landslide model with CAMELOT on high performance computers. Parallel Computing **29**(10) (2003) 1403–1418
6. Love, P.J., Nekovee, M., Coveney, P.V., Chin, J., González-Segredo, N., Martin, J.M.R.: Simulations of amphiphilic fluids using mesoscale lattice-Boltzmann and lattice-gas methods. Computer Physics Communications **153** (2003) 340–358
7. Talia, D.: Cellular processing tools for high-performance simulation. IEEE Computer **33**(9) (September 2000) 44–52
8. Hecker, C., Roytenberg, D., Sack, J.R., Wang, Z.: System development for parallel cellular automata and its applications. Fut. Gen. Comp. Sys. **16** (1999) 235–247
9. Guisado, J.L., Jiménez-Morales, F., Guerra, J.M.: Cellular automaton model for the simulation of laser dynamics. Physical Review E **67**(6) (2003) 066708
10. Guisado, J.L., Jiménez-Morales, F., Guerra, J.M.: Computational simulation of laser dynamics as a cooperative phenomenon. Physica Scripta **T118** (2005) 148–152
11. Guisado, J.L., Jiménez-Morales, F., Guerra, J.M.: Simulation of the dynamics of pulsed pumped lasers based on cellular automata. Lecture Notes in Computer Science **3305** (2004) 278–285
12. Worsch, T.: Simulation of cellular automata. Future Generation Computer Systems **16**(2-3) (1999) 157–170