Performance Analysis of a Parallel Discrete Model for the Simulation of Laser Dynamics

J.L. Guisado
Centro Universitario de Mérida, Universidad de Extremadura
Sta. Teresa Jornet, 38. 06800 Mérida (Badajoz), Spain
jlguisado@unex.es

F. Fernández de Vega
Centro Universitario de Mérida, Universidad de Extremadura
Sta. Teresa Jornet, 38. 06800 Mérida (Badajoz), Spain

K. Iskra
Section Computational Science
Faculty of Science, Universiteit van Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands

Abstract

This paper presents an analysis on the performance of a parallel implementation of a discrete model of laser dynamics, which is based on cellular automata. The performance of a 2D parallel version of the model is studied as a first step to test the feasibility of a parallel 3D version, which is needed to simulate specific laser systems. The 3D version will have to run on a parallel computer due to its runtime and memory requirements. The model has been implemented on a Beowulf Cluster using the message passing paradigm. The parallel implementation is found to exhibit a good speedup, allowing us to run realistic simulations of laser systems on clusters of workstations, which could not be afforded on an individual machine due to the extensive runtime and memory size needed.

1. Introduction

In the latest years, there has been an ever-increasing interest in the use of discrete models for the simulation of complex systems. Cellular automata (CA), in particular, are such a class of spatially and temporally discrete mathematical systems, characterized by local interaction and synchronous dynamical evolution [1]. They have provided good models for a wide variety of physical systems exhibiting cooperative phenomena [2], and have recently been successfully applied to model laser dynamics [3, 4, 5]. From a practical point of view, CA constitute a very interesting approach to carry out realistic high performance simulations, thanks to its intrinsic parallel nature which makes them particularly suitable to be naturally and efficiently implemented in parallel computers. Different parallel implementations of CA models[6, 7, 8, 9] and software tools for the programming of CA in parallel computers [10, 11] have been recently introduced.

In this work, we present an analysis on the performance of a parallel implementation in two dimensions of a discrete model of laser dynamics, based on CA. This model was presented in references [3, 4], and a further application of it was shown in reference [5]. While those papers discussed a sequential implementation of the model, to our best knowledge, this problem has never been modelled with a parallel CA. The performance of such a parallel implementation of the model is discussed in this work. Our aim is to study the possibility of running large size 2D simulations of this model in clusters of workstations. This result will be useful to test the feasibility of a parallel 3D version of the model, needed to make realistic 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. In Section 3 we describe the parallel implementation of the model. In Section 4 the performance of the implementation is analysed. Finally, the conclusions of this study are explained in Section 5.

2. Cellular automaton model

In this section, we summarize the CA model model originally presented in references [3, 4].

A cellular automaton defined on a two-dimensional square lattice of $N_c = L \times L$ cells with periodic boundary conditions is used to simulate a laser device. Two variables $a_i(t)$ and $c_i(t)$ are associated to 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 *Moore neighborhood* is considered, i.e. each cell has nine neighbors: the cell itself, its four nearest neighbors and the four next neighbors. The transition rules, which specify the state of a cell at time $t+1$ depending on the state of the cells included in its neighborhood at time t, determine the time evolution of the CA. These rules represent the different physical processes in a laser system at the microscopic level:

- Rule 1. Pumping: If the electronic state of a cell has a value of $a_i(t) = 0$ in time t, then in time $t+1$ that state will have a value of $a_i(t+1) = 1$ with a probability λ.
- Rule 2. Stimulated emission: If, in time t, the electronic state of a cell has a value of $a_i(t) = 1$ and the sum of the values of the laser photons states in the nine neighbor cells is greater than a certain threshold (which in our simulations has been taken to be 1), then in time $t + 1$ a new photon will be created in that cell: $c_i(t+1) = c_i(t) + 1$ and the electron will decay to the ground level: $a_i(t + 1) = 0$.
- 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 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

In this section, the parallel implementation of the model is summarized.

It is possible to obtain qualitative results which reproduce much of the laser phenomenology with a 2D automaton of moderate size. Nevertheless, if we want to obtain more detailed quantitative results, or if we want to carry out an extension of this model to 3D to simulate the specific behaviour of particular laser devices, a very large automaton has to be used. In this case, the algorithm will be time and memory consuming even for the most powerful sequential computers. Therefore, we have carried out a parallel implementation.

Parallelization was performed for distributed-memory MIMD systems using the message passing paradigm. The Parallel Virtual Machine (PVM) library [12] has been used, because we were interested in a further study of our model using dynamic load balance 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 was performed using the master-slave model. A separate control program (the “master”) is responsible for process spawning, initialization, collection of results, input/output of data and timing of functions. The “slave” programs, whose workloads are dynamically allocated by the master, perform the actual computations.

The computations involve three phases. The first one is the initialization phase, in which the master program performs the following operations: collection of input data from an external file (system size, number of partitions of the automaton, parameter values, number of time steps), spawning of slave programs, partitioning of the initial data of the automaton, sending of initial data and common information to each slave, collection of results from slaves for each time step, calculations with the whole results, output of final data to external files and timing functions to measure performance. The second phase, which is carried out by the slave programs, performs computation. In this phase, each slave computes the time evolution of the assigned partition of the automaton, following the CA evolution rules. After each time step, the values of the photon state $c_i(t)$ of the cells in the borders of each slave partition are communicated to the slaves dealing with the neighbour partitions. This communication is carried out directly between the slave programs. In addition, each slave computes the total number of electrons $a_i(t)$ and laser photons $c_i(t)$ in its CA partition after each time step and sends this information to the master, which can make some calculations with these results such as computing the Shannon’s entropy as described in [3]. The third phase is collection of final state and termination tasks. In this phase, the master optionally collects the final state of the CA partition from each slave, output final data to external data files, terminates the slaves execution and performs timing functions.

The cellular automaton is partitioned using a 1-dimensional domain decomposition, i.e. the CA is vertically partitioned in stripes, similarly as in [8]. Each subdomain is assigned to a different processor. This kind of partition has been used, instead of a 2-dimensional (checkerboard) domain decomposition, because it makes the communication structure simpler and minimizes the number of send/receive calls. On the other hand, the amount of data to be communicated in the 1D case is larger. Nevertheless, as shown in reference [6], a 1D decomposition is more favourable in runtime for a small to moderate number of nodes. For each subdomain, 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 neighbouring cells belonging to different subdomains.

4. Performance analysis

The simulations were performed in the Abacus high-performance computing cluster, a Linux Beowulf-style parallel cluster computer built by the Artificial Evolution Group (GEA) at the University of Extremadura (Spain). It runs the Rocks Linux cluster distribution, which allows a very scalable cluster management strategy [13, 14]. Abacus is a heterogeneous cluster composed of two different kinds of nodes: six nodes with an Intel Pentium-4 processor with 2.7GHz of clock frequency and four more nodes (including the master node responsible for the cluster coordination) with the same processor but with 1.8 GHz of clock frequency. Every node has 512MB of RAM memory. Communication between cluster nodes is carried out using a fast-ethernet 100 Mbps switch.

In order to test the performance of the parallel implementation, simulations have been carried out for different sizes of the cellular automaton grid (2520×2520, 1260×1260 and 630×630 cells) using a number of nodes ranging from 1 to 10. For each experiment, we have completed 1000 iterations (time steps) for a single value of the system parameters: $\lambda = 0.0125$, $\tau_c = 10$, $\tau_a = 180$. 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.

It must be noted that, as our cluster is heterogeneous, a particular configuration has been chosen for the experiments to avoid indeterminsm in the results. For the simulations with 1 to 6 nodes, the slave programs have been run on the "fast" (2.7GHz) machines. For simulations with 7 to 10 nodes, additional "slow" (1.8 GHz) machines have been used to complete the whole number of nodes. The master program was always run on the master node of the cluster (which runs at 1.8 GHz).

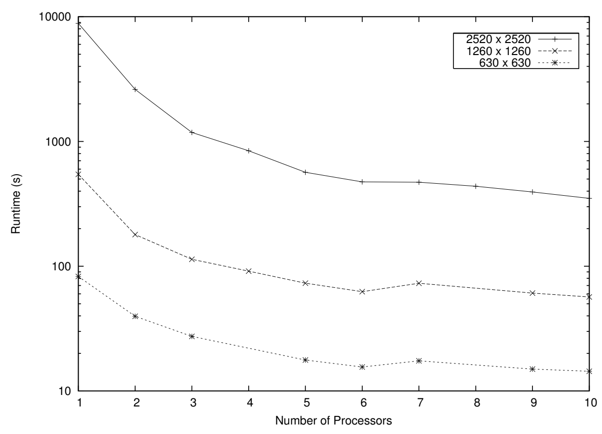

Figure 1. Runtime of the experiments, shown in a logarithmic scale, using three different system sizes, for different number of partitions of the whole CA. Each partition was run on a different processor.

In Fig. 1 the wall clock runtime of the experiments using three different system sizes, for different number of partitions of the whole CA (each one being handled by the slave program on a different processor) is shown, using a logarithmic scale. Runtime decreases with the number of processors, except when moving from 6 to 7 processors. The reason is that "fast" machines have been used for a number of processors from 1 to 6 and "slow" machines to complete a number of processors higher than 6. We must take into account that the CA application operates in lock-step mode, so the speed of the application is limited by the speed of the slowest running task. Therefore, adding one "slow" machine has the effect of forcing a slower speed for the whole application. In Fig. 1, there are no data points for the 630×630 data set with 4 and 8 processors and for the 1260×1260 data set for 8 processors. The reason is that in our parallel implementation of the CA laser model every partition must have exactly the same number of cells, so the width of the system (in cells) must divide the number of partitions used. As this is not obeyed in these cases, no data were taken for them. Nevertheless, enough data have been taken to be able to study the variation of the runtime with the number of processors, for different system sizes.

In order to measure the performance of the parallel implementation, it is interesting to study the speedup obtained for different number of partitions of the system running on different processors. The speedup of a parallel program is defined as the ratio of the runtime of the sequential version of the program to the runtime of the parallel version. We

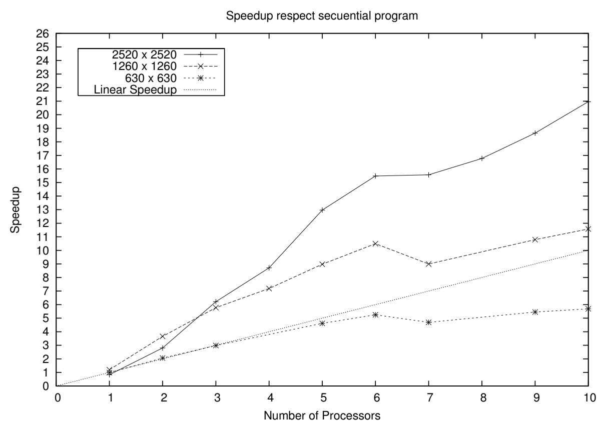

Figure 2. Speedup calculated relatively to the runtime of the sequential program.

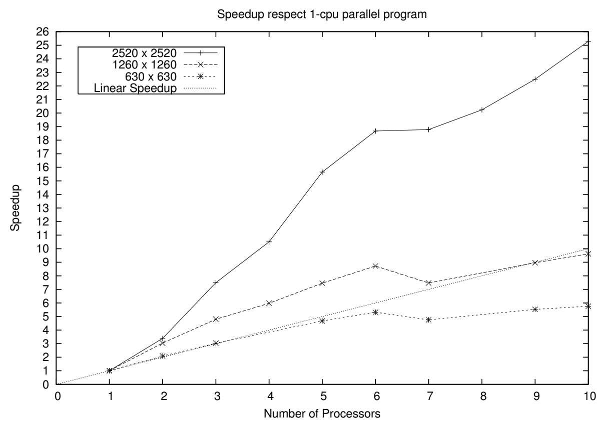

Figure 3. Speedup calculated relatively to the runtime of the parallel program for only one processor (i.e. using only one slave program containing the whole CA grid).

have calculated the speedup in two different forms. In the first one, the speedup has been calculated using the runtime of the sequential implementation of the CA model (i.e. the proper definition of the speedup has been used). In the second one, the speedup was calculated using the parallel program for only one processor (i.e. using only one slave program containing the whole CA grid), as is often done when lacking a sequential version of the program. The results are shown in Figs. 2 and 3, respectively. As expected, similar qualitative results are obtained, but differences can be observed in the actual speedup values. These differences are dependent on the size of the system: for a system size of 2520×2520 cells, the speedup is higher when calculated respect to the parallel program for only one processor, whereas for a smaller system size, the speedup is higher when calculated respect to the sequential version or there is no substantial difference. The reason of these differences is the efficiency of the parallel implementation (when run for only one node) in comparison to a purely sequential implementation, which is dependent on the size of the system. The latter is to be preferred, when available, to evaluate the performance of a parallel algorithm.

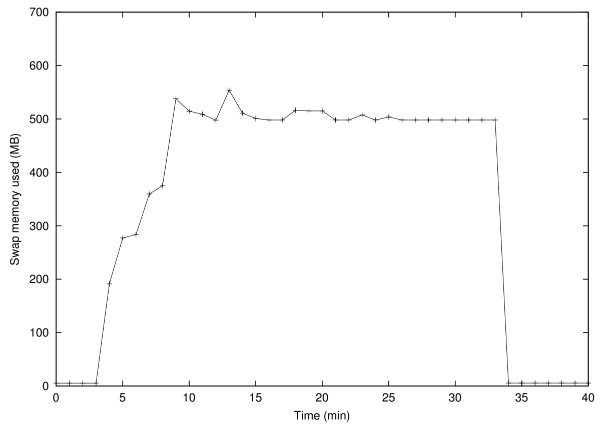

Figure 4. Amount of swap memory used by the slave node of the cluster during an execution of the parallel application for only one node, for a size of 2520×2520 cells.

Figs. 1 to 3 show that the implementation takes a good advantage of the parallelization: runtime decreases with the number of nodes and a good speedup is obtained, in spite of the fact that the border data have to be exchanged after every time step. Speedup increases with the number of nodes except in the jump from 6 to 7 nodes. As explained before for the runtime, the reason is that a heterogeneous cluster has been used, with "fast" machines for 1 to 6 nodes and "slow" machines to complete a number of nodes higher than 6.

Superlinear speedup is obtained specially for a system size of 2520×2520 cells, and also for 1260×1260 cells. In order to explain its source, the amount of swap memory used by the slave node in the execution of the parallel application for only one node has been studied for different system sizes. During the execution of the aplication with system sizes of 630×630 and 1260×1260, no swap memory is used by the system. On the other hand, for a system size of 2520×2520, the slave node needs to make extensive use of swap memory, as shown in Fig. 4. No swap memory is used by the slave nodes in the execution of the parallel application with the same system size for two slave nodes.

A very large system size will be needed in order to run a

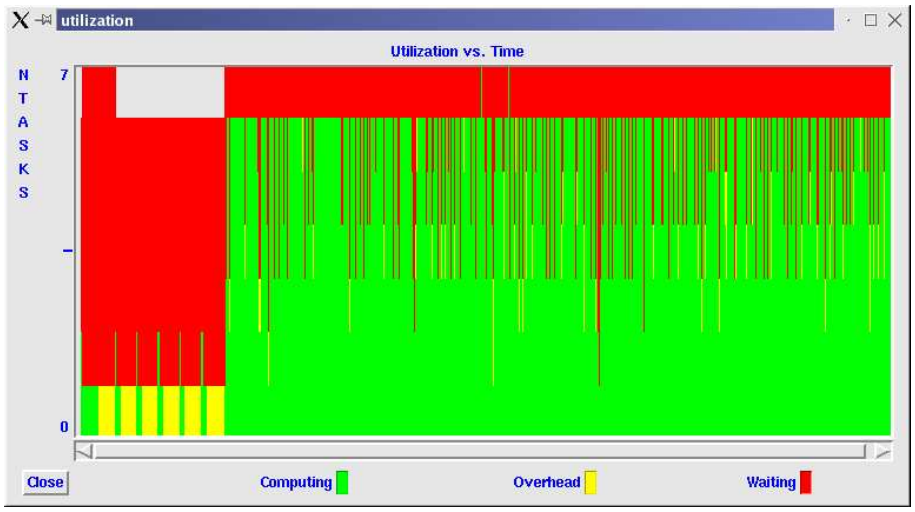

Figure 5. Gantt chart showing the different tasks executed versus time.

3D version of the model to simulate realistic laser devices. The runtime for such a large size would be intrinsically long. Furthermore, if a sequential version of that program would be executed in a single processor, the use of memory swapping would produce a serious additional increase in the runtime, making the calculation non-affordable. In contrast, if the parallel version of that program is run on the cluster using several nodes, the system is partitioned so the memory limitation can vanish. This fact, together with the parallel execution, can reduce the runtime and make the calculation affordable.

The execution of the parallel application has been analysed using XPVM, a graphical console for PVM which provides a performance monitor and debugger [15]. In Fig. 5, a Gantt chart showing the different types of tasks executed for each node versus time is shown. In this chart, tasks 1 to 6 correspond to the slave nodes and task 7 to the master node. The tasks marked as “Computing” are busy executing user computation, those marked as “Overhead” are busy executing PVM system calls and those marked as “Waiting” are idle waiting for messages. It can be observed that after the master process allocates the initial CA state to all the slaves, computation is the dominant task for slave processes for the rest of the execution, and the master process mostly waits.

In Fig. 6, a detail of the tasks executed by each node and the messages transferred between different nodes versus time, once the computation phase has started, is shown. Two distinct periods can be distinguished: computation and communication. In computation periods the CA state for the next time step is calculated by applying the evolution rules in each partition. In communication periods the photon state values of the cells in the borders of each partition are communicated to the slave nodes which handle the neighbouring partition, and the total number of electrons and photons in each slave partition are sent to the master node. As can be observed in Fig. 6, computation periods are much longer than communication periods. The average computation-to-communication ratio obtained for the parallel application for slave processes is of the order of 10. This rather high value indicates that the application is taking a good advantage of the parallelization.

5. Conclusions and future prospects

An analysis on the performance of a parallel implementation in 2D of a discrete model of laser dynamics based on cellular automata has been presented. This work serves as a first step to test the feasibility of a parallel 3D version of the model, which will be needed to make realistic simulations of specific laser systems. Such a 3D implementation would necessarily require a parallel implementation, due to its runtime and memory requirements.

In spite of the fact that some communication must be undertaken between the slaves and with the master after each time step, the implementation has been found to take a good advantage of the parallelization. Runtime decreases with the number of nodes and a good speedup is obtained. Superlinear speedup is found for large system sizes, due to memory swapping and finite memory effects. Therefore, the parallel implementation allows us to run realistic simulations on clusters of workstations, which could not be afforded on an individual machine due to the extensive runtime and memory size needed.

We expect to extend this work with the study of a parallel implementation of the model in 3D, which will be very

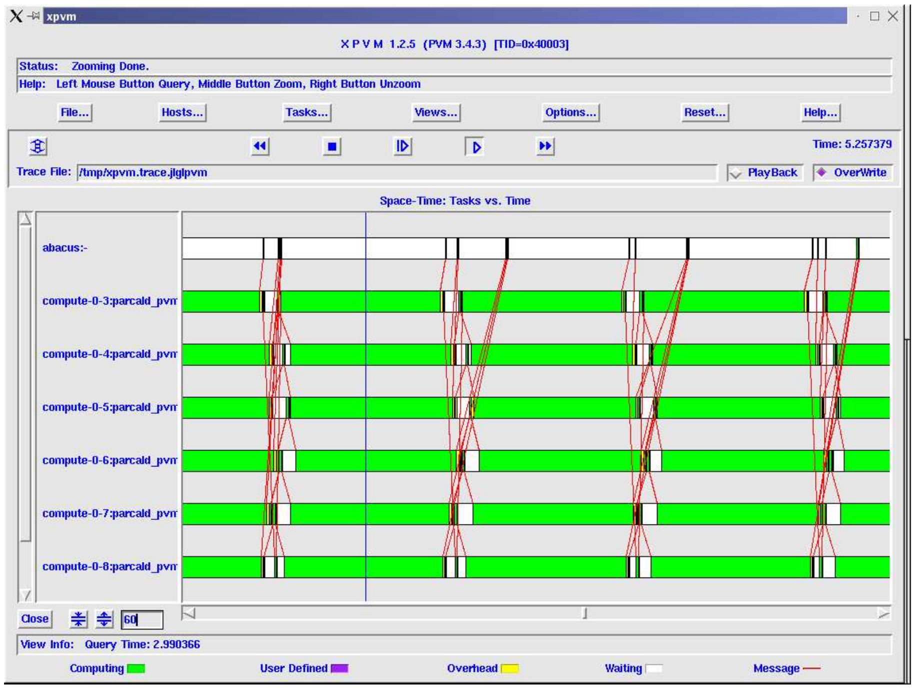

Figure 6. Gantt chart showing a detail of the tasks executed by each node and the messages passed between different nodes versus time, once the calculation phase has started.

useful to run realistic simulations. In addition, it would be interesting to investigate how this application benefits from dynamic load balance in heterogeneous non-dedicated parallel systems (usual target plattforms for this kind of application), and to study the scalability of the model for massive parallelism.

6. Acknowledgements

This work was supported by Project TRACER (TIC2002-04498-C05-01) of Ministerio de Educación y Ciencia, Dirección General de Investigación (Spain).

References

[1] J. von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, 1966.

[2] B. Chopard and M. Droz. *Cellular automata modeling of physical systems*. Cambridge University Press, 1998.

[3] J.L. Guisado, F. Jiménez-Morales, and J.M. Guerra. A cellular automaton model for the simulation of laser dynamics. *Physical Review E*, 67: 066708, 2003.

[4] J.L. Guisado, F. Jiménez-Morales, and J.M. Guerra. Computational simulation of laser dynamics as a cooperative phenomenon. *Physica Scripta*, T118: 148-152, 2005.

[5] J.L. Guisado, F. Jiménez-Morales, and J.M. Guerra. Simulation of the dynamics of pulsed pumped lasers based on cellular automata. *Lecture Notes in Computer Science*, 3305: 278-285, 2004.

[6] T. Worsch. Simulation of cellular automata. *Future Generation Computer Systems*, 16: 157-170, 1999.

[7] P.M.A. Sloot, J.A. Kaandorp, A.G. Hoekstra, and B.J. Overeinder. Distributed simulation with cellular automata: architecture and applications. In J. Pavelka, G. Tel, and M. Bartošek, editors, *SOFSEM'99: Theory and Practice of Informatics*, volume 1725 of *Lecture Notes on Computer Science*, pages 203–248, 1999.

[8] S Bandini, M. Magagnini. Parallel processing simulation of dynamic properties of filled rubber compounds based on cellular automata. *Parallel Computing*, 27: 643-661, 2001.

[9] P.J. Love, M. Nekovee, P.V. Coveney, J. Chin, N. González-Segredo, J.M.R. Martin. Simulations of

amphiphilic fluids using mesoscale lattice-Boltzmann and lattice-gas methods. *Computer Physics Communications*, 153: 340-358, 2003.

[10] D. Talia. Cellular processing tools for high-performance simulation. *IEEE Computer*, 33(9): 44–52, September 2000.

[11] C. Hecker, D. Roytemberg, J.-R. Sack, Z. Wang. System development for parallel cellular automata and its applications. *Future Generation Computer Systems*, 16: 235-247, 1999.

[12] A. Geist, A. Geguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam. *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*. The MIT Press, Cambridge MA, 1994.

[13] Rocks Cluster Distribution. http://www.rocksclusters.org

[14] P.M. Papadopoulos, M.J. Katz, G. Bruno NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters. *Concurrency and Computation: Practice and Experience*, 00: 1-20, 2002.

[15] XPVM: A Graphical Console and Monitor for PVM. http://www.netlib.org/utk/icl/xpvm/xpvm.html