Parallel Cellular Automata-based Simulation of Laser Dynamics using Dynamic Load Balancing

J.L. Guisado, F. Fernández de Vega, F. Jiménez Morales, Kamil Iskra and Peter M. A. Sloot

***Abstract*— In order to analyze the feasibility of executing a parallel bioinspired model of laser dynamics on a heterogeneous non-dedicated cluster, we evaluate its performance including artificial load to simulate other tasks or jobs submitted by other users. As the model is based on a synchronous cellular automaton (CA), using the SPMD (Single Program, Multiple Data) paradigm, it is not clear in advance if an appropriate efficiency can be obtained on this kind of platform. A dynamic load balancing strategy with two main differences from most previous implementations of CA based models has been used. First, it is possible to migrate load to cluster nodes initially not belonging to the pool. Second, a modular approach is taken in which the model is executed on top of a dynamic load balancing tool—the Dynamite system— gaining flexibility. Very satisfactory results have been obtained, with performance increases from 60% to 80%.**

I. Introduction

Cellular Automata (CA) are a class of spatially and temporally discrete mathematical systems characterized by local interaction and synchronous dynamical evolution [1]. They provide an excellent approach for modeling and simulating complex systems that has been used over the recent years in many fields of science and technology [2], [3]. A CA-based model for simulating laser dynamics was introduced in [4]. As shown there and in a recent review of the subject in [5], the model reproduces much of the phenomenology of laser systems and is an alternative to the standard modeling approach based on differential equations. This model can be very useful for situations such as lasers ruled by stiff differential equations, difficult boundary conditions, very small devices, etc. A parallel implementation of this model, necessary to carry out realistic 2D simulations of specific laser systems or for 3D simulations, was presented in [6]. In addition, it was found that the parallel implementation offers a good performance running on small dedicated computer clusters [7]. In this CA-based model of laser dynamics, information about the state of the cells included in the borders of the different partitions of the system must be exchanged after each time step, as represented in Fig. 1. This implies that it must be waited 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? In order to answer this question we have studied the efficiency of the algorithm on these conditions when using a dynamic load balancing strategy to optimize the use of the computing resources.

Sequential CA-based simulations can be used for education or for very simple systems, but 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 [8]. As CA normally operate in a lock-step mode, it is essential that some of the computing nodes are not overloaded with respect to the rest and thus dynamic load balancing is an important issue and has been previously studied in different forms. Several parallel implementations of CA focus on distributing the active cells between the nodes for CA in which some of the cells may become idle for a number of time steps [9], [10], [11] or on moving cells from heavily loaded nodes to more unloaded ones [12], [13], [14]. In other cases, the size of the partitions to be handled by each cluster node is adjusted, see for instance Refs. [15], [16]. These approaches can balance the load when some nodes are overloaded. But they are unable to migrate the jobs to new nodes which did not originally participate in the computation of the CA. This possibility would offer more flexibility for a real non-dedicated parallel computing environment and hence we have opted for using a dynamic load balancing approach that allows it in our present work.

Most of the parallel CA approaches directly implement the dynamic load balancing algorithm on the own CA algorithm. An example (in addition to most of the previously cited works) is offered by the P-CAM system [17], a special-

J.L. Guisado is with the 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. Fernández de Vega is with the Centro Universitario de Mérida, Universidad de Extremadura, Sta. Teresa Jornet, 38. 06800 Mérida (Badajoz), Spain.

F. Jiménez Morales is with the Departamento de Física de la Materia Condensada, Universidad de Sevilla. P.O. Box 1065, 41080 Sevilla, Spain.

Kamil Iskra is with the Argonne National Laboratory, Mathematics and Computer Science Division, 9700 South Cass Avenue, Argonne, IL 60439, USA.

Peter M. A. Sloot is with the Section Computational Science, Laboratory for Computing, System Architecture and Programming, Faculty of Science, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands.

This work has been supported by the NOHNES Project (TIN2007-68083-C02, Spanish Ministry of Science and Innovation), by the GRIDEX Project (PRI06A223, Consejería de Infraestructuras y Desarrollo Tecnológico de la Comunidad Autónoma de Extremadura, Spain) and by the Cátedra CETA-CIEMAT, Universidad de Extremadura.

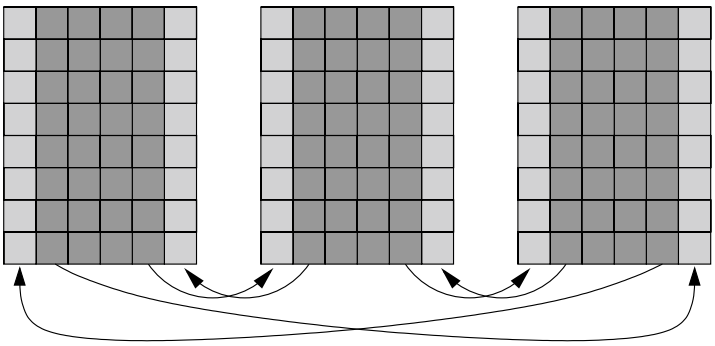

Fig. 1. In the parallel implementation of a CA, information of the state of the cells included in the borders of each partition of the system has to be communicated to the neighboring partition to be used in the computation corresponding to the next time step. In this example, the CA has been partitioned into parallel stripes. Each partition is assigned to a different processing node.

ized framework for parallel complex systems simulations which directly integrates data decomposition and dynamic load balancing into the framework functionality. Instead, we have chosen to execute the CA over a software tool which transparently implements the dynamic load balancing. This modular approach is more flexible, as changes can be introduced in the CA algorithm and in the dynamic load balancing algorithm without affecting each other. In addition, the load balancing tool can easily be changed. The tool that has been used is *Dynamite* [18], 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 [19], 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 [20] and [21] 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 maturity, flexibility and availability. However, other recent good dynamic load balancing systems which could also be used to execute this kind of simulations are the CAMELotGrid system [22], which is a specific tool to manage CA computations, and the general purpose framework designed by Vadhiyar and Dongarra, implemented and tested in the GrADS system [23]. An advantage of both systems over Dynamite is their possibility of integration on a grid computing environment. However, we decided to use a cluster computing environment because this is more adequate for the execution of a parallel CA (a *high performance computing* application) due to the low latency of the communications in comparison to a grid computing 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).

II. Simulation experiments

In the simulations a laser device is modeled by a two-dimensional, multivariable, partially probabilistic CA corresponding to a transverse section of the active medium in the laser cavity. The state of the CA cells represent electrons and laser photons and the set of transition rules defines the time evolution of the system. After specifying an initial state, the system is allowed to evolve for a number of iterations. In each time step, two macroscopic magnitudes (representing the total number of laser photons and of electrons in a particular energy state) are measured by counting the number of CA cells on a specific state. The response of the system depends on three parameters: the pumping probability (λ), the lifetime of photons (τ_c) and the lifetime of excited electrons (τ_a). A parallel implementation of this CA laser model for distributed-memory parallel computers using the message passing paradigm, discussed in [5], has been employed. A one-dimensional data decomposition is used in which the CA grid is vertically partitioned in stripes and each sub-domain is assigned to a different node. Two additional columns of *ghost cells* have been included at both sides of each sub-domain, as shown in Fig. 1, to store the state of neighboring cells belonging to different sub-domains.

The master-slave programming model has been used: a master program divides the CA grid in sub-domains and sends each to a slave program running on a different node, which calculates its time evolution. Algorithm descriptions of the master and slave programs are shown in Algs. 1 and 2, where operations involving communication between nodes have been indicated with a leading * character.

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

Algorithm 1 Pseudo code diagram for the master program

```
Input data
* Spawn slave programs
Partition the initial data of the automaton
* Send common information and initial data to each slave
for time step = 1 to maximum time step do
   * Collect partial results from slaves
   Perform intermediate calculations with partial results
   Output data to follow execution
end for
* Terminate slave programs
Perform final calculations
Output final results
```

Algorithm 2 Pseudo code diagram for the slave program

```
* Receive common information and initial data from master
* Exchange boundary cells state with slaves computing neighboring partitions
for time step = 1 to maximum time step do
   for each cell in the array do
      Apply transition rules
   end for
   Calculate populations after this time step
   Optional additional calculations on intermediate results
   * Send populations and other intermediate results to master
end for
```

message has arrived. Therefore, this exchange plays the role of a synchronization point between all the slave programs. This is illustrated in Fig. 2 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. More details and precise definitions of this parallel CA laser model can be found in Refs. [6], [7], [5].

In order to study the performance of the parallel application we have executed the same experiment under controlled conditions on the cluster, including artificial loads to simulate a normal non-dedicated cluster use. The particular experiment involved the computation of the time evolution of the system during $10,000$ time steps for a single value of the system parameters: $\lambda = 0.0125$, $\tau_c = 10$, $\tau_a = 180$. External load was simulated by a sequential C program with a simple assign instruction involving double precision numbers, inside a loop statement that iterates for a specified period of time. A similar procedure was used for example in [23]. The compilation of this program was carried out without any optimization to better obtain the desired loading result. In order to study the effect introduced by different levels of external load, the execution time was measured when running the artificial load on a number of cluster nodes ranging from 0 to 5, using both normal PVM or the modified PVM version included in the Dynamite load balancing system. 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 parallel CA application has been executed using 6 slave nodes plus a master node and a total of 10 nodes were available on the cluster. Inmediately after starting the CA application, the artificial load task has been initiated on a number of nodes, which range from 0 to 5 nodes and are always nodes to which one of the slave CA applications has been initially allocated also. The artificial load tasks kept on running for a time longer than the total execution time of the CA application.

III. Results and Discussion

The execution times and the improvement due to load balancing are presented in Table I. In the first row the execution time obtained without load balancing, using normal PVM, is shown. This is the same when running load on any number of nodes from 1 to 5. The reason is that the performance of the CA laser model application is limited by the performance of the slowest running task, because the system operates in a lock-step mode, as discussed in section I. In the following rows, it is shown the execution time when load balancing is employed (using Dynamite instead of normal PVM) and the artificial load is also run on a number of nodes ranging from 1 to 5. In addition, the

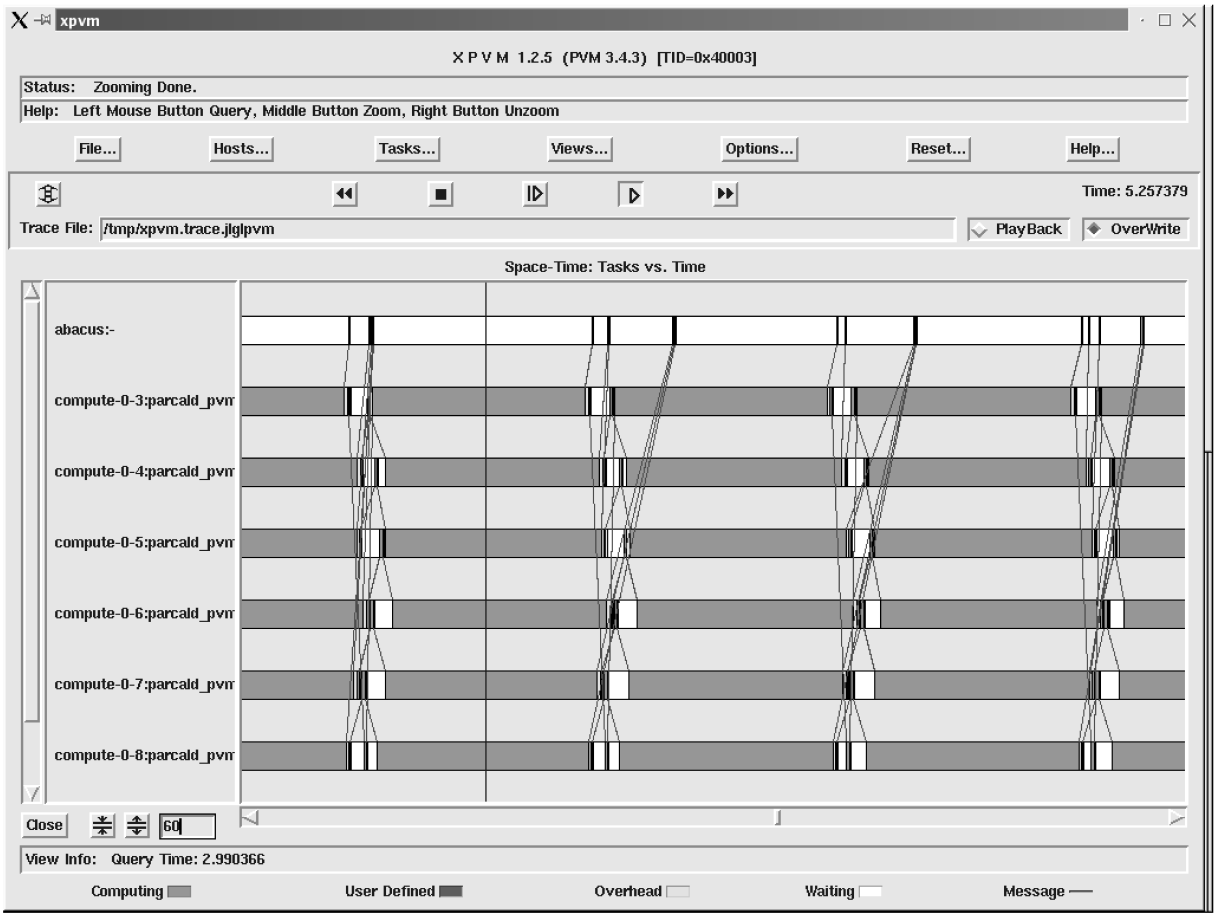

Fig. 2. Gantt chart depicting a detail of the tasks executed by each cluster node and the messages transferred between different nodes versus time, once the calculation has started. The exchange of neighboring states between nodes processing adjacent partitions at the beginning of each iteration acts as a synchronization point.

TABLE I

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	-

relative improvement between the execution time with and without (first row) load balancing is shown in the last column. 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.

An execution profile (number of executed time steps from the application versus time) is presented in Fig. 3. That gives an idea of the progression in the execution of the application. For this configuration (parallel application running on 6 slave nodes plus the master, on a cluster with 10 nodes) a very good improvement in the performance is obtained when running external load on up to 3 nodes. When there are idle nodes, the application takes a good advantage from the

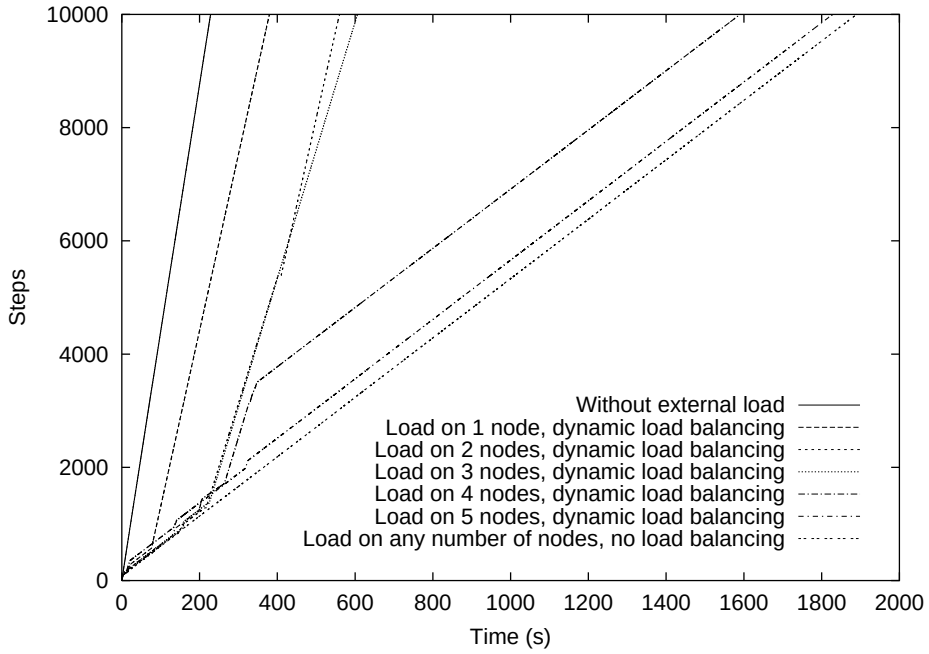

Fig. 3. Execution progress of the CA laser model application for different levels of artificial external load on the system. The system size was 840×840 cells. The number of cluster nodes used on the execution is 6.

dynamic load balancing, reducing the execution time by a factor of 3. When running external load on more than 3 nodes, a lower improvement is obtained but the execution time is still smaller than when no dynamic load balancing is used. 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 situation and performs the migration of some of the tasks of the system to balance the load. After that, for external load on a small number of nodes, the execution progress improves significantly, following a new straight line close to that one of the standard PVM. For external load on a higher number of nodes, in some cases the benefit obtained after migrations to try to balance the load on the system is very low and in other cases, after an advantageous migration of tasks, the dynamic load balancing system incorrectly migrates tasks again to let the system load unbalanced and obtain a sub-optimal execution progress. Another interesting result is that the dynamic load balancing system introduces 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. 3 (labeled as "Without external load") applies to both cases.

In order to study the effect of the system size on the performance of the application, simulations have been run for three different system sizes and the execution progress has been compared. The results are shown in Fig. 4. Relatively small CA sizes have been used in order to avoid the use of swap memory on the cluster nodes, that can happen for larger system sizes (as shown in [7]) and complicate the performance analysis. 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.

For the purpose of studying the regularity of the scheduling operation of the dynamic load balancing system, the application has been run under the same conditions for a number of times. In Fig. 5 the execution progress of four different runs of the same experiment with Dynamite are presented, showing some cases in which the load balancing system lets the load unbalanced and the execution time is not optimal. This also happened in the experiments reported in previous figures for a 10% - 20% of the executions, but these cases were not taken into account for the results presented. As migrations are not performed by the load balancing system in a very regular and deterministic way, it can be concluded that the scheduler component of the Dynamite load balancing system could be improved.

IV. Conclusions

We have studied the performance of a parallel discrete model of laser dynamics, based on a cellular automaton, running on a heterogeneous non-dedicated cluster using dynamic load balancing. Artificial external load has been included to simulate the effect of other tasks which can be running simultaneously on the cluster. 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. The potential problem of this kind of application on this environment is that all the computing nodes must have finished an iteration before the next one can be initiated. In order to obtain a good performance on a non-dedicated cluster where jobs from different users can be started on any computing node at any

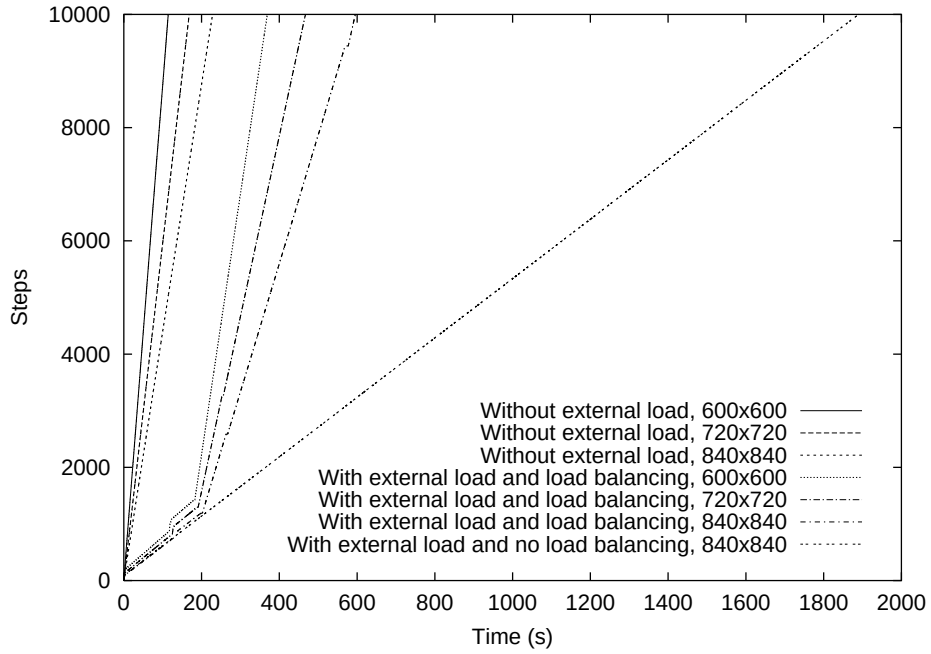

Fig. 4. Execution progress of the CA laser model application for different system sizes. The number of cluster nodes used on the execution is 6 and artificial external load has been run on 3 nodes.

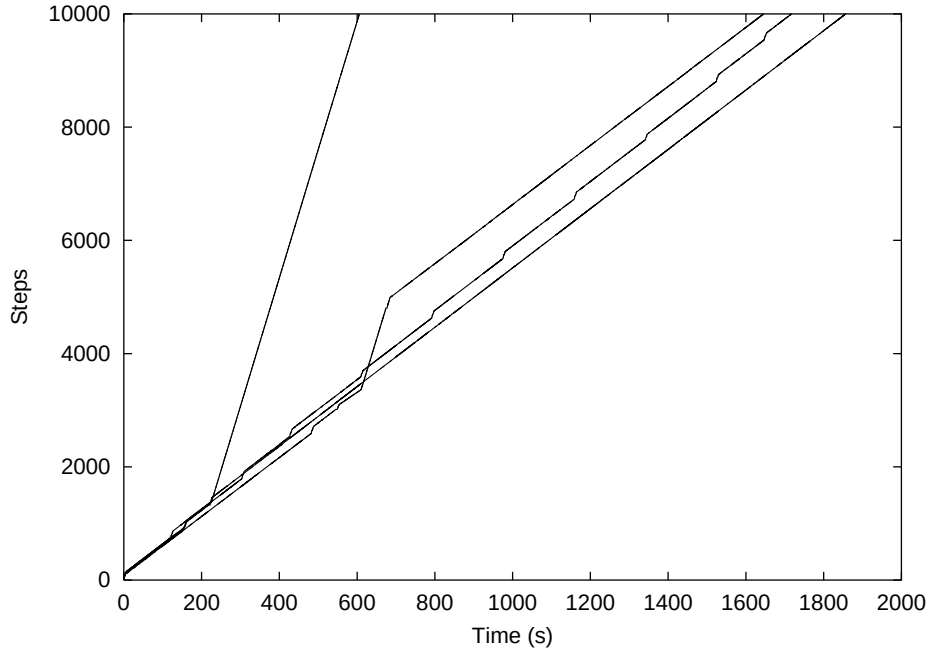

Fig. 5. Execution progress of 4 different runs of the application with Dynamite carried out under the same conditions. System size: 840×840 cells.

time, a dynamic load balancing strategy has been used, with two main differences with respect to most previous parallel CA implementations. First, we required that load could be migrated to new nodes initially not belonging to the pool. Second, the load balancing functionality has been uncoupled from the CA algorithm by running it on top of a dynamic load balancing software tool. Thus changes can be introduced to the CA algorithm or the dynamic load balancing strategy without disturbing each other. For this purpose, we have used Dynamite, an automated load balancing system that can migrate individual tasks which are part of a parallel program running with a message passing library—PVM in the current version. Very satisfactory results have been obtained: the load balancing strategy is able to improve the performance of the parallel application in levels from 60% to 80% when there are some idle nodes on the cluster to which some load

can be migrated. In all the studied cases, the execution time is always shorter than without the use of load balancing. However, the results indicate that the Dynamite system does not always choose the best configuration possible to balance the load, so further improvements can be introduced in the scheduler component of Dynamite. From the results, it can be concluded that it is feasible to execute this kind of algorithm on a heterogeneous non-dedicated cluster if using an adequate dynamic load balancing strategy. This ensures that a future 3D version of the laser CA model, which will necessarily have to be executed on a high performance parallel system, can have an appropriate efficiency on this environment.

REFERENCES

[1] A. Ilachinski. *Cellular automata. A discrete Universe*. World Scientific, Singapore, 2001.

[2] 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.

[3] Bastien Chopard and Michel Droz. *Cellular Automata Modeling of Physical Systems*. Cambridge University Press, Cambridge, 1998.

[4] J. L. Guisado, F. Jiménez-Morales, and J. M. Guerra. Cellular automaton model for the simulation of laser dynamics. *Physical Review E*, 67(6):066708, 2003.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] M. Cannataro, S. Di Gregorio, R. Rongo, W. Spataro, G. Spezzano, and D. Talia. A parallel cellular automata environment on multicomputers for computational science. *Parallel Computing*, 21(5):803–823, 1995.

[11] D. D'Ambrosio and W. Spataro. Parallel evolutionary modelling of geological processes. *Parallel Computing*, 33(3):186–212, April 2007.

[12] 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.

[13] G. A. Kohring. Dynamic load balancing for parallelized particle simulations on MIMD computers. *Parallel Computing*, 21:683–693, 1995.

[14] 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.

[15] 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.

[16] J. R. Weimar. Cellular automata for reaction-diffusion systems. *Parallel Computing*, 23(11):1699–1715, 1997.

[17] 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.

[18] 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.

[19] 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):101–105, 1996.

[20] 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.

[21] 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.

[22] G. Folino and G. Spezzano. An autonomic tool for building self-organizing grid-enabled applications. *Future Generation Computer Systems*, 23(5):671–679, 2007.

[23] S. S. Vadhiyar and J. J. Dongarra. Self adaptivity in grid computing. *Concurrency Computation Practice and Experience*, 17(2-4):235–257, 2005.