

Dynamical Load Balancing in Cellular Automata Models

Giuseppe Spingola¹, Giuseppe Zito¹, Donato D'Ambrosio¹,
William Spataro^{1*} and Rocco Rongo²

¹ Department of Mathematics and High Performance Computing Center,
University of Calabria, Italy

² Department of Earth Sciences and High Performance Computing Center,
University of Calabria, Italy

Abstract. Cellular Automata models are continuously gaining attention from the Scientific Community for their potentiality and efficiency in the modeling and simulation of complex systems. Here we present the last release of libAuToti, an open-source parallel library for implementing Cellular Automata based models. With respect to its former version, this release introduces a dynamic load balancing feature that can provide significant performance improvements when dealing with the simulation of topologically connected phenomena. In addition, the present version of the library permits 3D Cellular Automata model implementation and quiescent state management for optimization purposes. First results concerning load balancing experiments have demonstrated the usefulness of the feature in appreciably reducing execution times in comparison with its former, not-balanced, version.

Key words: Cellular Automata, Parallel Computing, Parallel Software Tools, Load Balancing, Modeling and Simulation

1 Introduction

Among different methodologies adopted in Computational Science, such as numerical analysis, high order difference approximations and finite differences, Cellular Automata (CA)[1] have proved to be a valid choice when the behavior of the system to be modeled can be described in terms of local interactions among its constituent parts. Well known field applications of CA are multiple, ranging from computational theory to physical and environmental modeling, from pedestrian and traffic simulation to cryptography. Refer, for instance, to the ACRI conference (e.g. [2]) series for up-to-date CA applications interested by the International Scientific Community.

In the context of geological phenomena modeling, Macroscopic Cellular Automata (MCA) [3] are particularly suitable for the modeling of spatial extended

* Corresponding Author: William Spataro, Department of Mathematics, University of Calabria, Via Pietro Bucci, I-87036 Rende, Italy; E-mail: spataro@unical.it.

systems whose dynamic can be described by local interactions at a macroscopic level. Well known examples of MCA applications include the simulation of lava [4] and debris flows [5], forest fires [6], besides many others.

In the framework of Cellular Automata development tools, libAuToti [9] represents an open-source, efficient and flexible solution, particularly suitable for the definition of MCA models. Similarly to the non-free Cellular Automata Simulation Environment CAMELot [13], libAuToti allows for a simple and concise definition of both the transition function and the other characteristics of MCA model definition. Furthermore, similarly to CAMELot, it was designed for both sequential and parallel execution, both on shared and distributed memory machines (thanks to the adoption of the Message Passing paradigm for the inter-processes communications), by hiding parallel implementation issues to the user.

An important issue in Parallel Computing applications is the distribution of computational load over the different processing elements, in order to exploit resources at best [11]. This is particularly relevant for MCA models of topologically connected phenomena like lava or debris flow in which the flow(s) can develop within a (small) sub-region of the cellular space during the simulation, depending on the topographic conditions and source location(s) [13]. If a typical data-parallel CA space allocation is adopted, where different CA space portions of the same size are assigned to different processing elements, this could represent a hitch, since most of the computation is taking place where “active” cells are located, overloading the related processing element(s). An efficient load-balancing technique is therefore often desirable. Accordingly, the version of libAuToti here presented introduces an automatic domain detection feature that dynamically balances computational load among processors.

In the following, MCA are briefly presented and the main characteristics of libAuToti summarily illustrated. Subsequently, the libAuToti Dynamical Load Balancing (DLB) feature is illustrated and results shown for a practical experiment carried out by considering the SCIDDICA MCA debris flow simulation model. A general discussion concludes the paper.

2 Macroscopic Cellular Automata

A Cellular Automata can be thought as a regular n -dimensional lattice of sites or, equivalently, as an n -dimensional space (called cellular space) partitioned in cells of uniform size (e.g. square or hexagonal for $n = 2$), each one embedding an identical finite automaton. The cell state changes by means of the finite automaton transition function, which defines local rules of evolution for the system, and is applied to each cell of the CA space at discrete time steps. The states of neighboring cells represent the input for the central one. The CA initial configuration is defined by the finite automata states at time $t = 0$. The global behavior of the system emerges, step by step, as a consequence of the simultaneous application of the transition function to each cell of the cellular space.

When dealing with the modeling of spatial extended dynamical systems, Macroscopic Cellular Automata (MCA) can represent a valid choice if the dy-

namics of such systems can be described in terms of local interaction at a macroscopic level. Well known examples of successful applications of MCA include the simulation of lava [4] and debris flows [5], forest fires [6], and highway traffic [14], besides many others. By extending the classic definition of Homogeneous CA, the use of a MCA model can facilitate the characterization of several aspects considered relevant for the correct simulation of the complex systems to be modeled. In particular, MCA provide the possibility to decompose the CA cell state in “substates” and allow the definition of “global parameters”. Moreover, the dynamics of MCA models (especially those developed for the simulation of complex macroscopic physical systems such as debris or lava flows) is often driven by the Minimization Algorithm of the Differences (cf. [3]), which translates in algorithmic terms the general principle for which natural systems leads towards the equilibrium condition. Refer to [3] for a more detailed description, besides theoretical aspects and some applications.

3 The libAuToti Macroscopic Cellular Automata library

MCA models of complex systems, as the SCIDDICA family (cf.[5]) for landslide simulation, often need to be the most possible efficient since, depending for example on the size of input data, simulations can even last days or actually weeks (cf.[15]). In these cases, the developer must implement proper optimization strategies (e.g. cf. [16]) or even parallelize the program (e.g. by means of MPI - Message Passing Interface). Different CA environments have been proposed, both for sequential and high performance parallel computers. Among parallel cellular systems, well known examples include Starlogo, P-CAM, PECANs, besides the above mentioned CAMELot. These environments are usually characterized by a CA-based programming language, an abstract Cellular Automata Machine for the execution of cellular programs, and a viewer for permitting the possibility to visualize cell states in a graphic format.

Though the above solutions can represent valid choices for the implementation of (parallel) MCA models, by providing important advantages like the use of a proprietary language, seamless parallel execution, a 2D/3D viewer (for the case of CAMELot), or even load balancing features (as in the case of CAMELot and P-CAM), major disadvantages include the non-extensibility of the software, lack of adequate debug facilities and reduced 3D visualization capabilities. At the contrary, libAuToti is not an integrated simulation environment, but an ANSI C++ (thus portable) library, which intends to offer the main features of CAMELot, as the possibility to simply define the model by ignoring low level details, to manage input and output data and to execute the simulation both sequentially or in parallel by adopting the Message Passing paradigm. A major advantage of the library is that the user does not need to concentrate in parallel issues, such as the CA space decomposition or message passing of boundary “ghost” cells between nodes: all these operations are transparently carried out by the library. Moreover, once the program has been compiled, the user has to simply decide how many processing elements the CA has to run on (e.g. `mpirun`

`-np 4 progname` for automatically executing `progname` in parallel on 4 processing nodes).

In `libAuToti`, the developer produces a standard C++ program by including the `libAuToti` library header file, with all the advantages that this approach permits. Among these, the possibility to choose the preferred development environment with the related debug facilities, and the possibility to easily introduce further features such as an optimization module or a 3D OpenGL viewer. Computational results of tests carried out on an Alpha Server SC45 supercomputer and on an Intel dual-core PC demonstrated the validity and efficiency of the library [9]. Refer to [9] for further details on CA modeling with `libAuToti`.

3.1 `libAuToti` Dynamic Load Balancing

In Parallel Computing, load balancing is referred as the technique to spread (or map) work between two or more processing elements, in order to get optimal resource utilization, and thus to reduce the overall computing time. In general, mapping techniques used in parallel algorithms can be classified into static and dynamic classes. It is worth to note that, the particular parallel programming paradigm and interactions among the concurrent tasks determines the suitability of a static or dynamic mapping. For instance, if the task size and/or size of data associated with the task are known a priori, a static mapping could represent the best choice. At the contrary, if task size and/or size of data are unknown, a static mapping can lead to a serious imbalance, making dynamic load balancing more effective. However, if the amount of data to be exchanged among processors gives rise to elevated communication times, this may outweigh the advantages of dynamic mapping. In this case, a static load balancing of task may be more suitable.

When dealing with parallel CA environments, a crucial task is the appropriate distribution of computational load (i.e. transition function computation for each cell) over the processing elements in order to exploit an effective use of resources [11]. This aspect is still more fundamental when simulating topologically connected phenomena where the set of “active” cells is limited to one of few domains [8]. For instance, in the case of a MCA lava flow model, a typical simulation is originated from few active cells (i.e., the crater cells) to further increase in number as the simulation spreads in the CA space, thus overloading the processing element(s) where active cells are located.

As stated previously, when dealing with CA, a parallelism known as *data parallelism* represents the natural type of decomposition [13]. In `CAMELot`, a static load balancing strategy based on the scattered decomposition technique [7] is considered, which effectively can imply the reduction of the number of stationary state cells (i.e. whose state does not change at the next step of computation) involved in the calculation. Therefore, similarly to `CAMELot`, `libAuToti` adopts a simple data decomposition partitioning, where the CA space is simply subdivided in N row-wise strips (C++ stores data in a row-wise manner), where N denotes the number of processors on which the CA runs. This kind of distribution requires each processor P_i to communicate only to processors $P_i - 1$ and

$P_i + 1$, so that a simple topology is obtained. This simplification has proved to be efficacious [13], since communication occurs only between adjacent processors, permitting a good advantage for those simulations in which the phenomenon evolves over one or few topologically connected domains, as may be the case of a lava or debris flow.

Both static and dynamic load balancing approaches have been adopted in parallel CA frameworks. Experiments carried out on CAMELot implementations of MCA models using its load balancing feature resulted in better execution times on Transputer-based networks [8], while the advantage considerably decreases on present-day High Performance Computers or Clusters. Even other CA environments adopt solutions for proper work balance. For instance, a dynamic load balancing solution has been adopted in P-CAM [12]. P-CAM (Parallel Cellular Automata Modeling) is a simulation environment based on a computational framework of interconnected cells (virtual particles). The cells of a Cellular Automata are arranged in graphs that define the cell interconnections and interactions. At the beginning of a simulation, a decomposition of these graphs on a parallel machine is generated and a load balancing strategy is used to migrate cells.

P-CAM allows to achieve better performance on modern parallel machines with respect to CAMELot but, unfortunately, it is less suitable for the implementation of MCA models. We therefore propose an alternative dynamic load-balancing algorithm, implemented in the last release of *libAuToti*, consisting in an automatic feature for the migration of cells from higher loaded processors to lesser ones. The originality and, by some sense, congenial approach that has been devised for the algorithm, is that it is based on the Minimization Algorithm of Differences [3], which is at the basis of several MCA models. Even if major specifications can be found in [4], we briefly describe the algorithm in the next section.

The Minimization Algorithm of the Differences In general, the dynamics of many natural phenomena can be modeled in terms of flows of certain quantities at a local level [3]. In the context of the MCA approach this is accomplished by the application of the Minimization Algorithm of the Differences. This computes flows from the central cell towards its adjacent ones in order to minimize unbalance conditions in the neighborhood, leading to the local equilibrium. It is based on the following assumptions:

- two parts of the considered quantity must be identified in the central cell: these are the unmovable part, $u(0)$, and the mobile part, m ;
- only m can be distributed to the adjacent cells; let $q_o(x, y)$ denote the flow from cell x to cell y ; m can be written as:

$$m = \sum_{i=0}^{\#X} q_o(0, i)$$

where $q_o(0, 0)$ is the part which is not distributed;

- the quantities in the adjacent cells, $u(i)(i = 1, 2, \dots, \#X)$ are considered unmovable;
- the flow from the central cell towards the i th neighbor is computed as the difference between $u(i)$ and the last computed average value a :

$$q_o(0, i) = \begin{cases} a - u(i) & i \in A \\ 0 & i \notin A \end{cases}$$

Note that the simultaneous application of the minimization principle to each cell gives rise to the global equilibrium of the system. The correctness of the algorithm is stated in [3], i.e. it minimizes equation (1). Eventually, a relaxation rate, $r_r \in [0, 1]$, can be introduced, denoting that the equilibrium conditions may not be reached in a single CA step; the obtained values of outflows are therefore multiplied by r_r (if $r_r=1$, no relaxation is induced; if $r_r=0$, there will be no outflows towards the neighborhood).

3.2 The DLB Algorithm

The proposed DLB algorithm, based on the Minimization Algorithm of the Differences, considers three factors for achieving an optimal load balancing result. As noticed in typical dynamical load balancing techniques, a frequent movement of data among processors can lead to a serious degradation in overall parallel computation. A parameter, P_s , is therefore considered, specifying the number of CA steps after which a load balancing operation can be accomplished. The second factor that is considered in the algorithm is that load migration can occur only if the unbalance between neighboring processors, in terms of execution time, is greater than a given threshold, P_t . This prevents data exchange among processors when unbalance conditions are tolerable. The last factor that the algorithm provides for is the possibility to relax data exchange between processors (determined by the application of the Minimization Algorithm), in order to prevent excessive communication that could give rise to performance degradation. The parameter P_r , corresponding to the Minimization Algorithm's relaxation rate [3], is therefore considered in the *libAuToti* load balancing algorithm.

With these assumptions, the balancing algorithm considers a 1-dimensional CA-like array, having as many cells as the number N of processing elements on which the CA is running, and neighborhood consisting of the adjacent left and right cells, besides the central one. No wraparound condition is considered, since strips 0 and $N - 1$ do not interact in the *libAuToti* decomposition scheme. The value of the j th cell of the array represents the j th processor time taken after P_s computational steps. After each P_s CA steps, normal computation stops (i.e. CA evolution is temporarily halted) and, under the condition that the local temporal unbalance is greater than P_t , the Minimization Algorithm is applied to the cells of the 1-dimensional CA-like array.

As a result, for each cell of the array, temporal outflows $\tau(0, i)$ are computed. Here, the notation $\tau(0, i)$ denotes a flow from the central cell (indexed by 0) to the i th adjacent one. Note that, since such temporal outflows are determined

by the Minimization Algorithm, their “distribution” reduces the unbalance conditions between adjacent processors. Such result is achieved indirectly by the *libAuToti* balancing algorithm. If R_j denotes the CA rows associated to the j^{th} processor and t_j the corresponding computational elapsed time, the load balancing algorithm computes the flows of rows, $\rho(0, i)$, to be effectively distributed to adjacent processors, proportionally to the temporal outflows $\tau(0, i)$, as follows:

$$\rho(0, i) = \left\lfloor (R_j - 1) P_r \frac{\tau(0, i)}{t_j} \right\rfloor$$

Note that at most $R_j - 1$ rows can be distributed, since at least one row must remain for each processor. Once that all outflows are computed, all processors migrate their boundary rows towards neighbor ones, permitting to redistribute the computational load efficiently for the next P_s CA steps.

The pseudo-code of the DLB algorithm, applied to a CA evolving for S steps, is specified in the following pseudo-code:

```
libAuToti_DLB_Algorithm (Ps, Pt, Pr) {
    repeat {
        while [NOT((CA_step MOD Ps) == 0)]
            proc_tot_time += proc_time
            NORMAL_EXECUTION
            CA_step ++
        SEND_TIMINGS_TO_PROC_0
        IF (my_rank==0)
            BUILD_1D_CA_LIKE_ARRAY()
            MIN_ALG(CA_LIKE_ARRAY, Pt, Pr)
            SEND_DLB_INFO()
        EXCHANGE_ROWS
    until [CA_step==S]
}
```

where:

- NORMAL_EXECUTION is the code representing the normal CA execution;
- SEND_TIMINGS_TO_PROC_0 sends each processor time (no communication time) to processor 0;
- BUILD_1D_CA_LIKE_ARRAY() represents the code for the construction of the 1D CA-like array;
- MIN_ALG(CA_LIKE_ARRAY,Pt,Pr) is the application of the Minimization Algorithm of Differences to the 1D CA-like array consisting in the various processor timings;
- SEND_DLB_INFO() is the code representing the sending of rows information to neighboring processors, as determined by the application of MIN_ALG(CA_LIKE_ARRAY,Pt,Pr);
- EXCHANGE_ROWS is the effective row exchange among processing elements, as determined by the Minimization Algorithm.

As it can be seen, the DLB algorithm works jointly with the normal libAuToti CA code. At the beginning, a normal (not load balanced) evolution of the CA is executed for the first P_s steps. When P_s steps are completed, all processors send their timings to processor 0 which, in a typical master-slave processing scheme, collects them and, if significant unbalance conditions are detected, constructs the 1D CA-like array where cells represent the timings of processors after P_s steps. At this point, normal CA execution is halted and processor 0 applies the Minimization Algorithm in order to minimize timings between processors, computing the number of rows that the processor has to transfer to the appropriate neighboring ones in order to decrease its computational load. After the appropriate amount of rows is computed, this information is sent to each processor, after which, effective rows are sent to neighboring processors.

3.3 Experimental Results

A set of experiments, executed for evaluating the performance of the libAuToti DLB algorithm, were carried out by considering the SCIDDICA MCA debris flow model. A grid of 296 columns 420 rows, representing the DEM (Digital Elevation Model) of the Tessina landslide, occurred in Northern Italy in 1992, was considered. DEM's cells have 10 m side. In order to unbalance processing elements, landslide sources were placed in the upper rows of the morphology, at higher elevations. As the simulation is carried out, the landslide expands to lower topographic altitudes, thus progressively interesting other processing elements. Simulations performed by the former version of the libAuToti library, regarding landslide CA modeling, can be found in [10].

Table 1. Execution times of experiments carried out for evaluating the performance of the libAuToti Dynamic Load Balancing algorithm by considering the SCIDDICA CA debris-flow model

	Without DLB (s)	With DLB (s)	Improvement (%)
Core 2 Duo PC (2 cores)	3980	3102	22%
Xserve Beowulf (8 procs)	1152	955	17%
Alpha Server SC45 (16 procs)	107	80	25%

Experiments were performed on three different machines, namely a standard Intel T7700 2.4GHz Core 2 Duo PC, a Beowulf cluster composed by 4 Apple XServe G5 2GHz bi-processor nodes, and a 16 processor HP Alpha Server SC45. In particular, the latter high performing computer is composed of 4 SMP nodes, each with 4 Alpha 64-bit 1250 Mhz EV68 processors and 16 MB cache. In all cases, the DLB algorithm was applied with parameters $P_s = 1000$, $P_t = 0.001$ and $P_r=0.6$. This choice determined a total of 10 applications of the DLB algorithm during the simulation (the simulation was performed over 10000 CA steps), together with a moderate balancing rate (to not overload the Gigabit interconnection network of the Beowulf cluster). As regards the experiments

carried out on the standard dual-core PC, the application of the DLB algorithm allowed an improvement of the 22% of the overall execution time (about 3102 s against 3980 s of the not-balanced algorithm) while, the application of the DLB algorithm allowed an improvement of the 16% on the Beowulf cluster (about 872 s against 1039 s of the not-balanced algorithm). The best results were obtained for the Alpha Server SC45 supercomputer, where a 25% improvement of was obtained for the experiments (about 80 s against 107 s of the not-balanced algorithm). Table 1 summarizes the obtained results.

4 Conclusions

We presented the latest release of the libAuToti C++ parallel free library for Macroscopic Cellular Automata, which introduces a dynamic load balancing feature to better exploit computational resources and reduce overall execution time. The algorithm automatically performs load balancing among processors in order to minimize processor timings, according to an algorithm based on the Minimization Algorithm of the Differences, widely used in diverse MCA models of complex systems. Experiments, referred to the MCA SCIDDICA model for landslide simulation and executed for evaluating the advantage of the dynamically load balanced version of the library with respect the previous non-balanced one, evidenced significant improvements on a standard dual-core based PC, a Beowulf cluster with Gigabit Ethernet interconnection network, and an Alpha SC45 supercomputer. Best results, of about the 25%, were achieved on this latter. Nevertheless, performance improvement of about the 16% was also achieved on the Beowulf cluster. Although smaller, this must be considered a good result in consideration of the presence of a slow network (in terms of latency) interconnecting the four processing nodes. Future testing will be done in order to compute most favorable DLB parameters and further MCA models and/or CA configurations will be considered, besides performing experiments on other parallel machines. Still, the use of an automated optimization technique (like an evolutionary algorithm or an ad hoc heuristics), will help in calibrating DLB parameters with respect to both the adopted parallel machine and the implemented particular simulation model.

Acknowledgments. A special thanks goes to Prof. Salvatore "Toti" Di Gregorio, Prof. Gino Mirocle Crisci, Dr. Maria Vittoria Avolio and Dr. Valeria Lupiano for the common researches. The libAuToti library is freely downloadable at the URL <http://autoti.mat.unical.it>.

References

1. von Neumann, J.: Theory of self reproducing automata. University of Illinois Press, Urbana (1966)

2. VV.AA. 8th International Conference in Cellular Automata for Research and Industry In: H. Umeo, S. Morishita, K. Nishinari, T. Komatsuzaki and S. Bandini (eds.) ACRI 2008. LNCS, vol. 5191. Springer, Berlin (2008)
3. Di Gregorio, S., Serra, R.: An empirical method for modelling and simulating some complex macroscopic phenomena by cellular automata. *Fut. Gen. Comp. Sys.* 16, 259–271 (1999)
4. Crisci, G.M., Di Gregorio, S., Rongo, R., Spataro, W.: The simulation model SCIARA: the 1991 and 2001 at Mount Etna. *J. Vul. Geo. Res* 132, 253–267 (2004)
5. Di Gregorio, S., Rongo, R., Siciliano, S., Sorriso Valvo, M., Spataro, W.: Mount Ontake landslide simulation by the cellular automata model SCIDDICA-3. *Phy. Chem. Ear., Part A.* 24, 97–100 (1999)
6. Trunfio, G.A.: Predicting Wildfire Spreading Through a Hexagonal Cellular Automata Model. In: P.M.A. Sloot, B. Chopard and A.G. Hoekstra (eds.) ACRI 2004. LNCS, vol. 3305, pp. 725–734. Springer, Berlin (2004)
7. D.M. Nicol, J.H. Saltz, An analysis of scatter decomposition. *IEEE Trans. Comp. C* 39, 1337–1345 (1990)
8. Di Gregorio, S., Rongo, R., Spataro, W., Spezzano, G., Talia, D.: High performance scientific computing by a parallel cellular environment. *Fut. Gener. Comp. Sys.* 12, 357–369 (1997)
9. Spingola, G., D’Ambrosio, D., Spataro, W., Rongo, R., Zito, G.: Modeling Complex Natural Phenomena with the libAuToti Cellular Automata Library: An example of application to Lava Flows Simulation. In: Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (2008)
10. Spataro, W., D’Ambrosio, D., Spingola, G., Zito, G., Rongo, R.: libAuToti, A Parallel Cellular Automata Library for Simulation: An example of Application to Landslides. In: Proceedings Summer Computer Simulation Conference (2008)
11. Grama, A., Karypis, G., Kumar, V., Gupta, A.: An Introduction to Parallel Computing: Design and Analysis of Algorithms, Second Edition. Addison Wesley (2003)
12. Schoneveld, A., de Ronde, J.F.: P-CAM: a framework for parallel complex systems simulations. *Fut. Gener. Comp. Sys.* 16, 217–234 (1999)
13. Cannataro, M., Di Gregorio, S., Rongo, R., Spataro, W., Spezzano, G., Talia, D., A parallel cellular automata environment on multicomputers for computational science. *Paral. Comput.* 21, 803–824 (1995)
14. Di Gregorio, S., Festa, D.C., Rongo, R., Spataro, W., Spezzano, G., Talia, D.: A microscopic freeway traffic simulator on a highly parallel system. *Parallel Computing: State-of-the-Art and Perspectives*, In: E.H. D’Hollander, G.R. Joubert, F.J. Peters and D. Trystam (Eds.), *Advances in Parallel Computing* 11, pp. 69–76 (1996)
15. D’Ambrosio, D., Rongo, R., Spataro, W., Avolio, M.V., Lupiano, V.: Lava Invasion Susceptibility Hazard Mapping Through Cellular Automata. In: S. El Yacoubi, B. Chopard, and S. Bandini (eds.) ACRI 2006. LNCS, vol 4173, pp. 452–461. Springer-Verlag, Berlin (2006)
16. Walter, R., Worsch, T. Efficient Simulation of CA with Few Activities. In: P.M.A. Sloot, B. Chopard and A.G. Hoekstra (eds.) ACRI 2004 LNCS, vol 3305, pp. 101–110. Springer-Verlag, Berlin (2004)