The Journal of Supercomputing (2023) 79:9273–9287
https://doi.org/10.1007/s11227-023-05043-3

Tailoring load balancing of cellular automata parallel execution to the case of a two-dimensional partitioned domain

Alessio De Rango[1] · Andrea Giordano[2] · Giuseppe Mendicino[1] · Rocco Rongo[3] · William Spataro[3]

Accepted: 2 January 2023 / Published online: 10 January 2023

Abstract
In this paper, techniques for dynamic load balancing of the cellular automata parallel execution are presented for the case of domain space partitioned along two dimensions. Starting from general closed-form expressions that allow to compute the optimal workload assignment in a dynamic fashion when partitioning takes place along only one dimension, we tailor the procedure to allow partitioning and balancing along both dimensions. Both qualitative and quantitative experiments are carried out that assess performance improvement in applying load balancing for the case of two-dimensional partitioned domain, especially when the load balancing takes place along both dimensions.

Keywords Load balancing · Parallel computing · Cellular automata

✉ William Spataro
william.spataro@unical.it

Alessio De Rango
alessio.derango@unical.it

Andrea Giordano
giordano@icar.cnr.it

Giuseppe Mendicino
giuseppe.mendicino@unical.it

Rocco Rongo
rocco.rongo@unical.it

[1] Department of Environmental Engineering (DIAm), University of Calabria, Ponte Bucci, 87036 Rende, CS, Italy

[2] Institute of High Performance Computing and Networking (ICAR), National Research Council (CNR), Ponte Bucci, 87036 Rende, CS, Italy

[3] Department of Mathematics and Computer Science (DeMACS), University of Calabria, Ponte Bucci, 87036 Rende, CS, Italy

1 Introduction

The simulation of complex systems is a very compute intensive task that can strongly benefit from the support of modern parallel computer systems. Cellular automata (CA) have proven their suitability for systems whose behaviour can be described in terms of local interactions [1]. CA were studied by John von Neumann to study self-reproduction problems [2] and have been developed by numerous researchers and applied in both theoretical and scientific fields ([3–8]). Due to their local and independent rules, complex systems simulations can be easily implemented on parallel machines.

Parallel computing [9] has undoubtedly proved its effectiveness in many application scenarios (e.g. [10]). Nevertheless, overhead can arise due to the parallelization process itself, which can reduce the obtainable benefits ([11–14]). This is due to the fact that individual processing elements cannot carry out their computation in isolation since parallel activities must exchange data during the computation (e.g. the halo exchange in CA parallel execution - [15]). This need for synchronization, together with an unbalanced workload assignment, can strongly affect the overall parallel execution time. Though CA can benefit from local synchronization [16] (i.e. each processing element requires to exchange halo borders only with *neighbour* processing elements), computational performances in CA parallel execution can be degraded especially when the dynamics of the modelled phenomenon is confined in a sub-region of the entire cellular space and further expands during the simulation [17]. As a consequence, an effective load balancing technique can mitigate this problem.

In parallel computing, load balancing (LB) [18] is referred to as the technique for properly partitioning the computation between the processing elements of a parallel computer, in order to obtain an optimal use of resources, with the purpose of reducing the overall execution times. In this paper, we present a dynamical LB procedure in the case of two-dimensional cellular automata partitioned along both X and Y dimensions, where load balancing can take phase either on one or both dimensions. Our work extends [19], where an analytical procedure that allows to calculate the optimal assignment of the workload, referred to mono-dimensional space partitioning, is presented. In particular, the closed-form expressions of [19] are purposely exploited for the case of two-dimensional partitioning. The paper is organized as follows. In Sect. 2, the parallel execution of CA models is discussed. In Sect. 3, the analytic procedure for the optimal workloads assignment for the case of mono-dimensional CA partitioning is summarized. In Sect. 4, two kinds of two-dimensional load balancing techniques are presented. In Sect. 5, experimental results, referred to both a qualitative and quantitative analysis, are shown. Eventually, conclusions and future developments are given in Sect. 6.

2 Parallel execution of cellular automata

Cellular automata (CA) can be easily adopted to model and simulate complex systems characterized by a high number of interacting elementary components. Thanks to their implicit parallel nature, CAs can be productively parallelized across multiple

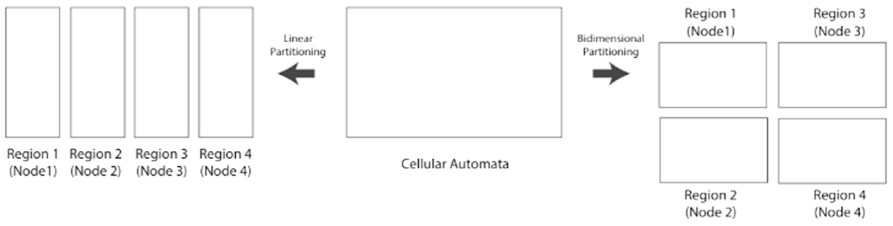


Fig. 1 The cellular space partitioned into regions which are associated with parallel computing nodes. Two alternative types of partitioning are shown, mono-dimensional and two-dimensional

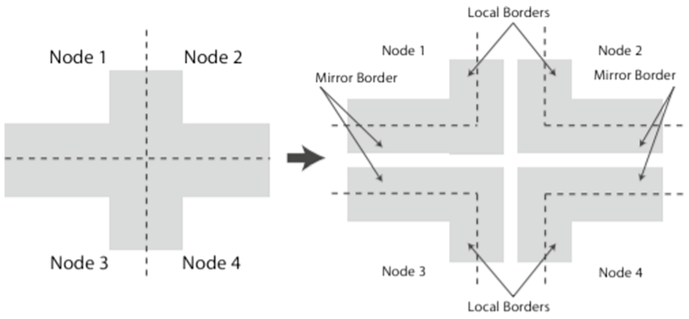


Fig. 2 Border areas of four adjacent nodes in the case of two-dimensional partitioning

parallel machines to scale and speed up their execution. Execution of CA on both sequential and parallel computers consists in a step-by-step evaluation the transition function for each cell of the cellular space.

The parallelization of CA execution can be efficiently achieved by partitioning the initial cellular space into different regions (or territories), which are assigned to the different processing elements (node) (e.g. [12, 20, 21]), as shown in Fig. 1. Each node is in charge of executing the transition function of all the cells belonging to the region it manages. Since for any cell the computation of the transition function depends on the state of its neighbourhood, in order to keep parallel execution consistent, the states of these boundary cells (often called *halo* cells in the CA literature) must be exchanged between neighbouring nodes at each computation step. As seen in Fig. 2, the border area of a region (the halo cells) is divided into two different sub-areas: the local border and the mirror border. The local border is managed by the local node, and its content is replicated in the mirror border of the adjacent node.

3 Dynamic load balancing of cellular automata

Load balancing techniques [22] can be mainly classified in *static* or *dynamic* based on whether the computation load is statically distributed to processing elements before execution, or where it is dynamically assigned during the parallel execution. When the distribution of the computational load is known *a priori* or can be easily predicted, static LB may be the most appropriate choice. Nevertheless, as for the case of CA modelling of natural phenomena, the evolution of the workload is unknown before execution, and a static mapping can lead to a critical imbalance, thus leading to the need of a dynamic load balancing.

In general, the LB phase does not take place at each step but is carried out at a predefined step rate or when a significant unbalanced condition occurs. An analytical procedure is presented in [19], which describes how the load balancing phase is achieved for mono-dimensional partitioning. In particular, the goal is to achieve a perfectly balanced execution among adjacent nodes by keeping their so-called *step times* as uniform as possible, where a *step time* is defined as the time needed by a node to execute one computational step. At each step, step times are collected and stored by each node and then sent to a specific *master* node that is in charge of establishing the optimal workload exchanges among nodes during the LB phase.

Let us consider a set of N nodes/regions arranged in a linear topology where each node has only one near node on the left and only one near node on the right, as shown in Fig. 3. Let S_i and T_i, with $0 \leq i < N$, be the size (i.e. the number of columns) and the step time of region i, respectively. The load balancing problem can be stated as determining, for given $(S_i, T_i), 0 \leq i < N$, the values of workload exchanges $\Delta x_i, 0 \leq i < N - 1$.

Δx_i values can be computed through formulas taken from [19], and in particular by expression (1) in which:

$$\widehat{\Delta x}_i = \frac{t_i}{i+2} + \sum_{j=i+1}^{N-2} \frac{(i+1)t_j}{(j+1)(j+2)}$$

$$t_i = -(i+1)\hat{S}_{i+1} + \sum_{j=0}^{i} \hat{S}_j$$

The formulas above have been obtained by considering a different LB problem, linked to the original one, where the mathematical formulation turns out to be more tractable and consists in a linear system that requires few algebraic manipulations in order to be solved in $O(N)$ steps, where N is the number of computing nodes (cf. [19] for details).

In the next section, we show how to exploit these formulas to the case of two-dimensional CA partitioning.

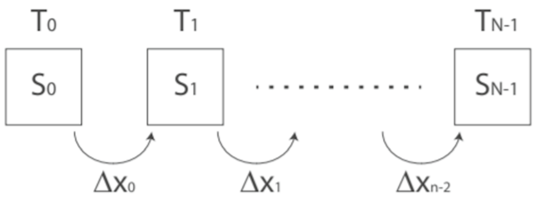


Fig. 3 Load balancing scheme for CA partitioned and balanced along the x dimension. For each node i, S_i represents the size (i.e. the number of columns in this case), T_i the experienced step time and Δx_i the workload exchange between nodes i and $i + 1$

$$\Delta x_i = \begin{cases} c_i \widehat{\Delta x}_i \\ \text{ if } \widehat{\Delta x}_i \geq 0, \widehat{\Delta x}_i \leq \hat{S}_i \\ \sum_{j=i-k}^{i} c_j \hat{S}_j + c_{i-k-1} \left(\widehat{\Delta x}_i - \sum_{j=i-k}^{i} \hat{S}_j \right) \\ \text{ if } \widehat{\Delta x}_i \geq 0, \widehat{\Delta x}_i \geq \sum_{j=i-k}^{i} \hat{S}_j, \widehat{\Delta x}_i < \sum_{j=i-k}^{i} \hat{S}_j + \hat{S}_{i-k-1}, k \geq 0 \\ c_{i+1} \widehat{\Delta x}_i \\ \text{ if } \widehat{\Delta x}_i < 0, -\widehat{\Delta x}_i \leq \hat{S}_{i+1} \\ -\left(\sum_{j=i+1}^{i+k+1} c_j \hat{S}_j + c_{i+k+2} \left(-\widehat{\Delta x}_i - \sum_{j=i+1}^{i+k+1} \hat{S}_j \right) \right) \\ \text{ if } \widehat{\Delta x}_i < 0, -\widehat{\Delta x}_i \geq \sum_{j=i+1}^{i+k+1} \hat{S}_j, \widehat{\Delta x}_i < \sum_{j=i+1}^{i+k+1} \hat{S}_j + \hat{S}_{i+k+2}, k \geq 0 \end{cases} \tag{1}$$

4 Load balancing of two-dimensional partitioned cellular automata

In this section, we show the implementation of the LB procedure referred to the case of CA partitioned along both x and y dimensions. In particular, we present the case where the LB phase takes place only along one dimension (i.e. the x dimension) and the one where LB is performed along both dimensions.

In two-dimensional partitioning, the cellular space is partitioned in N_x and N_y nodes along the x and y dimensions, respectively. As a consequence, the nodes are specified by a pair (x, y) and the sizes and step time are indicated as $S^x_{x,y}$, $S^y_{x,y}$ and $T_{x,y}$, with $0 \leq x < N_x$ and $0 \leq y < N_y$, where S^x and S^y denote the size along x and y dimensions. Given that $S^x_{x,y}$ is invariant with respect to y, we can indicate this quantity simply as S^x_x. In other words, S^x_x is the size of the x^{th} partition along the x dimension. The same applies also for $S^y_{x,y}$, which can be replaced by S^y_y. Figure 4 shows the local/mirror borders exchange that now occurs between each node and its 8 neighbours.

The formulas outlined in Sect. 3 can be exploited also for the two-dimensional partitioned case, but need to be adapted to this scenario. In particular, the formulas are now computed for each x and y load balancing separately. For the load balancing of along the x dimension, the (S_i, T_i) values of the previous section are here replaced by $\left(S^x_i, \sum_{y=0}^{N_{y-1}} T_{i,y}/N_y \right)$, that is we take the x size and the average of the step times of the nodes belonging to the ith partitioning along x. The LB phase along the y dimension is carried out analogously.

4.1 Load balancing along one dimension

In this section, we present the LB procedure for two-dimensional CAs partitioned only along one dimension, let us say x (see Fig. 5).

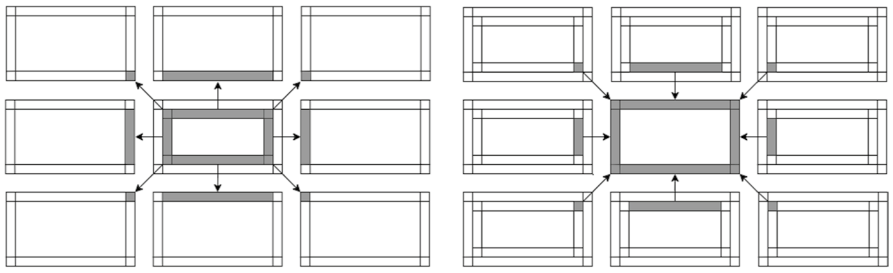


Fig. 4 *Local/mirror border* exchanges in two-dimensional partitioning. On the left, local borders of the central nodes are copied to the mirror borders of the neighbours. On the right, the local borders of the neighbours are copied to the local border of the central node

In order to simplify the message exchange scheme, two phases are envisioned: in the first, the usual border exchange is carried out, while the effective workload exchange takes place in a second phase. This choice was driven by the complexity of performing the workload exchange in a single phase. In particular, diagonal exchanges turn out to be quite *"tricky"*, due to the fact that both x workload exchange and y border exchange have to be considered at the same time. In our approach, the diagonal workload exchange is avoided by exchanging also the y mirror border when the x workload exchange takes place. For example, let us consider the workload exchange between node A and B in Fig. 6 (the same applies to nodes C and D). Differently from a typical two-dimensional neighbour data exchange (see Fig. 4) the transferred data (the grey part) also include the y mirror border of the portion of columns to be transferred. After the exchange, node B will host a coherent portion of its mirror border without retrieving it from node C. The correctness of the approach is guaranteed by the border exchange carried out in the first phase, which ensures mirror borders are already updated before being included in the subsequent workload exchange. The advantage of this approach appears to be more significant when the load balancing is carried out also for the y dimension, as in the case detailed in the next section.

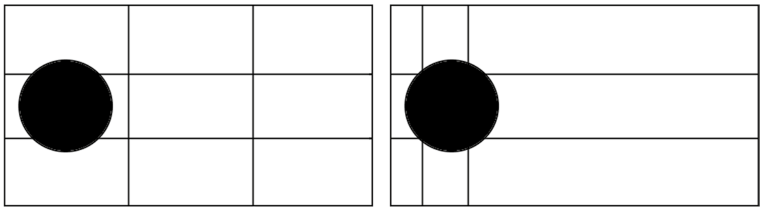


Fig. 5 One-dimensional load balancing. The figure shows the configuration of the CA model before and after the LB phase

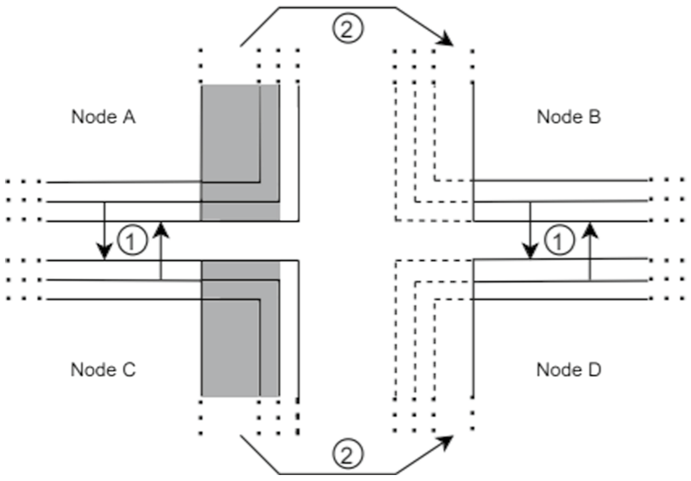


Fig. 6 Adjacent nodes workload exchange along the x dimension including y mirror borders. In the first phase (1), the borders are exchanged along y dimension, while, in the second phase (2), the workload (highlighted in grey) is actually exchanged along x dimension

4.2 Load balancing along both dimensions

In this section, we present the LB methodology application also for the case when workload is exchanged along both x and y dimensions. The effect of this two-dimensional load balancing is shown in Fig. 7.

The comparison between Figs. 5 and 7 clearly suggests that performing both x and y LB results in a more fair workload distribution and thus leading to better overall performances, as shown in the following Experimental Results section. Analogously to the one-dimension LB case, the LB is performed in more than one phase. Specifically, the first phase is devoted to the border exchange, the second one concerns the x workload exchange, while the last one regards workload distribution along the y dimension.

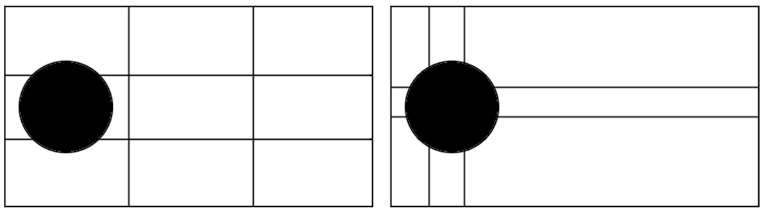


Fig. 7 Two-dimensional load balancing. The figure shows the configuration of the CA model before and after the LB phase

5 Experimental results

In this section, we summarize the results of the carried out experiments in order to evaluate the effectiveness of the proposed load balancing approaches. Two sets of experiments were envisioned aiming to test the goodness of these approaches from both *qualitative* and *quantitative* point of view.

All the experiments were carried out on a workstation composed of 2 nodes, each equipped with a 20 core/40 threads Intel(R) Xeon(R) E5-2650 v3 CPU 2.30 GHz and with 32 GB RAM. The MPI technology was used for message exchanges among processing elements. As a measure of performance, speed-up rates have been calculated and compared for three different scenarios: (i) non-balanced parallel execution, (ii) LB applied on x dimension and (iii) LB applied on both x and y dimensions.

For the qualitative analysis, the CA model consists of a simple *ball* moving throughout the CA space, while a computational fluid dynamics model, namely the *SciddicaT* landslide CA model, is exploited so as to carry out the quantitative performance analysis for a real-case test-bed scenario.

5.1 Qualitative experiments: the moving ball CA model

The *ball* CA model consists of a set of *active* and *inactive* cells. Active cells are those cells falling inside a given ball area, while inactive cells are the remaining ones. The initial configuration of this CA model is portrayed in Fig. 8.

At each time step, the transition function is designed in order to move the ball diagonally through CA space. In addition, the active cells transition function also executes a certain "dummy" computation so as to increase the computational load for these cells. In such a way, the considered model execution is characterized by a "graphically identifiable" computational load moving across the CA domain during the execution advancement, thus giving rise to the dynamic load unbalance conditions required for properly assessing the effectiveness of the proposed LB techniques.

The configuration parameters adopted in this first set of experiments are detailed in Table 1.

At the beginning of the CA execution, the CA space is equally split among nodes as seen in Fig. 8. Just after the first LB phase takes place, the new space partitioning, for the two considered LB approaches, turns out to be the one shown in Fig. 9a and b, respectively.

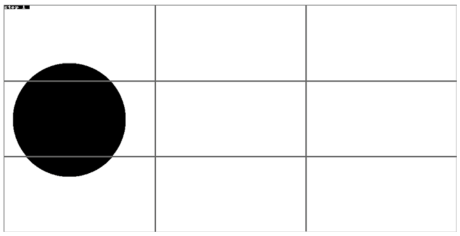


Fig. 8 Initial configuration of the *ball* CA model

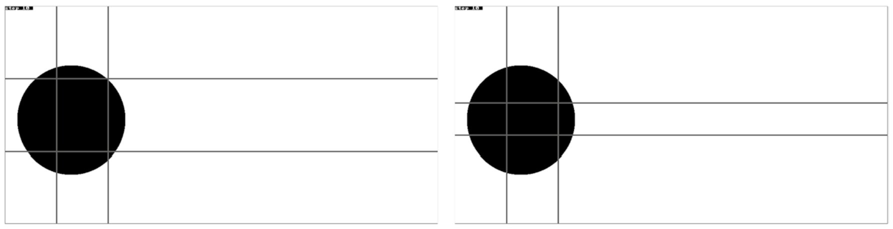
Table 1 Configuration parameters of the qualitative experiment

Parameter	Value
Total steps	400
LB step rate	10
MPI processes	9
CA size	500×1000 cells
Ball radius	125 cells

Afterwards, the second LB phase execution produces the partitioned schema to be as in Fig. 10a for the LB taking place only for the x dimension, and in Fig. 10b for the case when LB is performed for both dimensions. As one can notice, this new space partitioning adequately adapts to the new ball position.

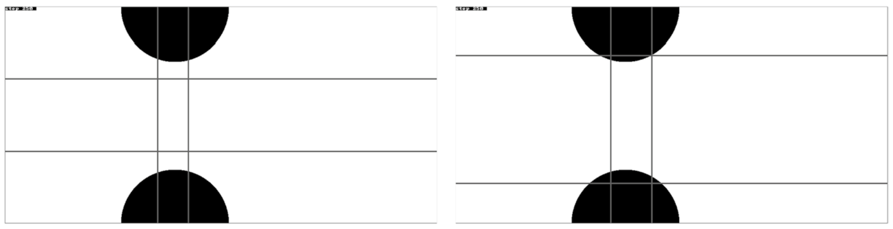
As it can noted from the previous figures, both LB procedures turn out to distribute the workload quite fairly among the processing elements, while the second LB strategy, namely balancing along both dimensions, turns out to be even more effective. For example, looking at Fig. 10a we can see a case in which balancing only along x dimension does not achieve a fully balanced condition as the three central nodes do not have any portion of the ball assigned to them. This is not the case for the approach in which both x and y are balanced, as it can be seen in Fig. 10b.

A similar balancing behaviour can be observed also after the last LB phase as shown in Fig. 11a and b,



(a) LB only along x dimension (b) LB along both x and y dimension

Fig. 9 The *Ball* CA model and space partitioning after the first LB step



(a) LB only along x dimension (b) LB along both x and y dimension

Fig. 10 The *Ball* CA model and space partitioning after the second LB step

5.2 Quantitative experiments: the SciddicaT CA model

As previously stated, a second set of experiments is devoted to quantitatively analyse the benefits of the proposed LB by measuring the improvements achieved in terms of execution time reduction. In these tests, we adopt a real-case testbed consisting of the SciddicaT CA debris flow model [23].

5.2.1 The SciddicaT CA model

SciddicaT is a simplified landslide model able to simulate the dynamics of a generic fluid-type flow over a real topographic surface. The experiments have been carried out on a real-case scenario representing the DEM (Digital Elevation Model) of the Tessina landslide, which occurred in Northern Italy in 1992 ([23]). Figure 12 shows the landslide covering area (highlighted in light grey). The landslide source, i.e. the area where the landslide was triggered, is represented by the area highlighted in dark grey which corresponds to the higher topographic elevation of the landslide. During the event, the landslide expands to lower topographic altitudes, thus progressively interesting all the area coloured in grey. As evident, the CA model of this landslide event can be a good candidate for suitably testing a load balancing approach. Indeed, the cells interested in the landslide vary during the simulation, thus producing a dynamic load unbalanced condition.

5.2.2 Experimental results

In this section, we report the results referred to a second set of experiments for assessing the performance improvement. The main configuration parameters of these experiments are shown in Table 2.

Figure 13 shows speedup curves for the three scenarios of interest, i.e.: (i) for the standard non-balanced execution, (ii) for the case in which the LB is applied only on the *x* dimension and (iii) for the case when both *x* and *y* load balancing takes place. As one can notice, the achieved results confirm what expected, namely, both LB strategies exhibit performance advantages with respect to the non-balanced execution and, in addition, the strategy of balancing on both dimensions clearly outperforms the one dimension LB strategy.

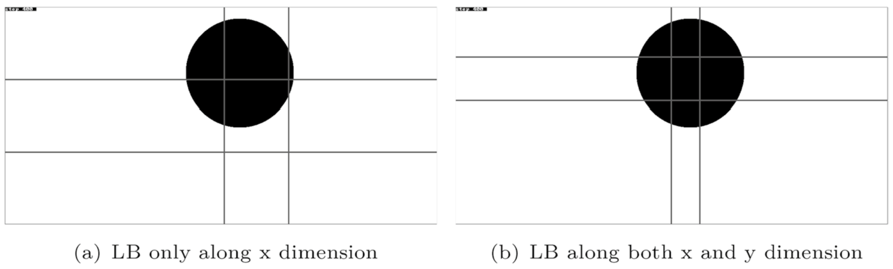

(a) LB only along x dimension
(b) LB along both x and y dimension

Fig. 11 The *Ball* CA model and space partitioning after the last LB step

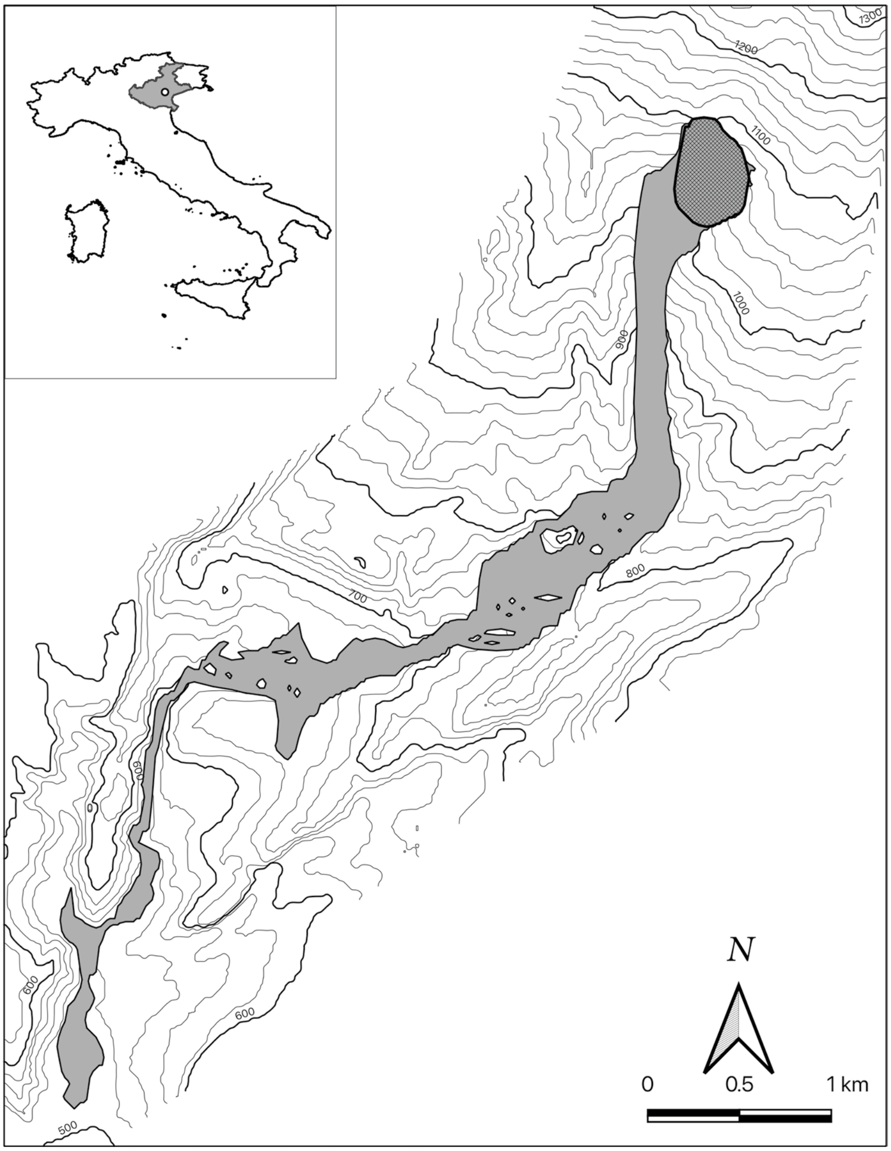


Fig. 12 The 1992 Tessina landslide topographic map. The grey colouring represents the area affected by the landslide. The darker grey area refers to the landslide source

Table 2 Configuration parameters of the SciddicaT model experiments

Parameter	Value
Total steps	4000
LB step rate	32
MPI processes	1–36
CA size	566×732 cells

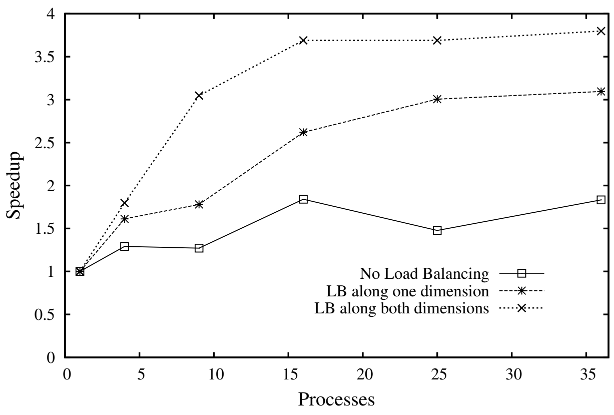


Fig. 13 Speed-up comparison between both types of load balancing strategies and standard parallel executions

By looking at Fig. 13 there are other two interesting observations to make:

- The trend of the curves is a bit *"oscillating"* and results in being a bit different with respect to an ideal speed-up curve. This phenomenon is due to the model-specific load distribution across the CA space. In this context, indeed, the parallel performance can be quite sensitive to the adopted space partitioning schema based on how this latter adapts to the specific model load distribution. In other words, given a partitioning, the more loaded portions of space could fall only in some nodes, leaving the others completely unloaded, or can be quite equally distributed across the nodes. This trend behaviour is quite evident looking at the non-balancing case, since the partitioning remains fixed during the entire execution. In fact, as we move from a fixed partitioning to a more properly load-balanced one, the trend becomes smoother and smoother, as expected for typical speed-up curves.
- The achieved speed-up values turn out to be quite modest compared to the theoretical achievable speed-ups. This is due, again, to the specific load distribution of the Sciddica model and is particularly significant for the non-balanced scenario. As a consequence, rather than considering the mere speed-up values, it can be more interesting to show the percentage of improvement of the LB strategies with respect to the non-balanced execution. To this aim, Fig. 14 witnesses significant improvements due to the considered load balancing techniques especially for the case of both dimensions load balancing.

A last set of experiments has been carried out in order to evaluate and compare the burden related to communication and synchronization. The aim is to confirm that the communication burden is much lower with respect to the synchronization burden and can be considered negligible in the majority of real-world application scenarios. In particular, we set up a configuration of 16 nodes (4×4) using the not balanced version, the only-X balanced version and the both-X-and-Y balanced version. We

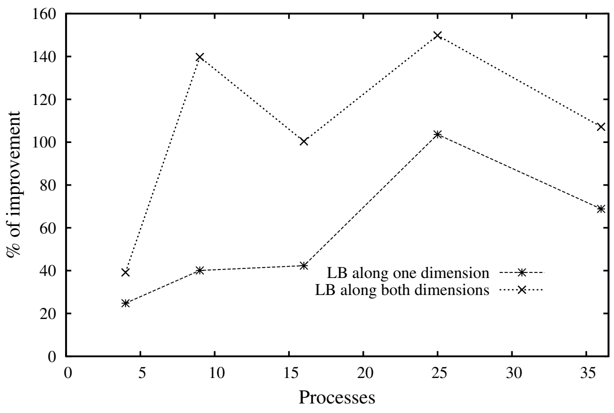


Fig. 14 Percentage improvement of both types of load balancing strategies w.r.t. the standard parallel execution

profiled the execution by simply recording the duration times for the MPI asynchronous send/receive operations (in order to estimate the communication burden) and the actual transition function computation for all the cells (in order to estimate the effective computation time). The total parallel execution times have been recorded as well. The ratio between the communication burden and effective computation results in being 0.2% for all the 3 versions, whereas the ratio of the synchronization burden and the effective computation results 28.6% for the non-balanced version, 19.3% for the only-*X* balanced version and 18.2% for the both-*X*-and-*Y* balanced version. These simple experiments allow us to conclude that: (i) the communication burden can be considered negligible with respect to the synchronization burden, as expected, (ii) the load balancing advantages are confirmed by the reduction of the synchronization burden, and (iii) the 18.2% ratio for the both-*X*-an-*Y* balanced version suggests that there is still room for further improvements in the load balancing algorithm.

6 Conclusions

In this paper, we focused on the load balancing of the parallel execution of cellular automata when the cellular space is partitioned along two dimensions. In particular, we present and compare two possible strategies: the first referred to the case of load balancing performed along only one of the two partitioning dimensions, while the second referred to the case when the load balancing is applied along both dimensions. Two sets of experiments have been carried out. The first set of experiments aims to assess the performance improvement from a qualitative point of view, by showing how the CA space partitioning dynamically adapts to changes in the model computational load. The second set of experiments was based on a real testbed model, namely the SciddicaT landslide CA model, in order to qualitatively assess the performance improvement in terms of speed-up. All experiments have confirmed performance improvements for both strategies with respect to non-balanced parallel execution. In addition, as expected, balancing along both x and y dimensions results in an even better speedup improvement.

Future work is geared to improve the load balancing technique for the two-dimensional partitioned CA domain, by introducing a more fine-grained load balancing strategy where we abandon a grid-like space partitioning, thus obtaining a higher degree of freedom in partitioning the space.

Author's contributions All authors have equally contributed to the research and contents of the manuscript. The first draft of the manuscript was written by AG and WS, and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Funding Open access funding provided by Università della Calabria within the CRUI-CARE Agreement. The authors have no relevant financial or non-financial interests to disclose. All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

Data availibility Not Applicable.

Declarations

Conflict of interest The authors have no competing interests to declare that are relevant to the content of this article.

References

1. De Rango A, Furnari L, Giordano A, Senatore A, D'Ambrosio D, Spataro W, Straface S, Mendicino G (2021) Opencal system extension and application to the three-dimensional richards equation for unsaturated flow. Comput Math Appl 81:133–158. https://doi.org/10.1016/j.camwa.2020.05.017
2. von Neumann J (1966) Theory of self-reproducing automata. University of Illinois Press, Champaign, IL, USA
3. Wolfram S (1984) Universality and complexity in cellular automata. Phys D 10:1–35
4. Aidun CK, Clausen JR (2010) Lattice-Boltzmann method for complex flows. Annu Rev Fluid Mech 42:439–472
5. Ntinas VG, Moutafis BE, Trunfio GA, Sirakoulis GC (2016) Parallel fuzzy cellular automata for data-driven simulation of wildfire spreading. J Comput Sci 21:469–485
6. De Rango A, Furnari L, Giordano A, Senatore A, D'Ambrosio D, Straface S, Mendicino G (2020) Preliminary model of saturated flow using cellular automata. In: Sergeyev YD, Kvasov DE (eds) Numerical computations: theory and algorithms. Springer, Cham, pp 256–268
7. Furnari L, Senatore A, De Rango A, De Biase M, Straface S, Mendicino G (2021) Asynchronous cellular automata subsurface flow simulations in two- and three-dimensional heterogeneous soils. Adv Water Resour 153:103952

8. Renc P, Pecak T, De Rango A, Spataro W, Mendicino G, Was J (2022) Towards efficient GPGPU cellular automata model implementation using persistent active cells. J Comput Sci 59:101538. https://doi.org/10.1016/j.jocs.2021.101538
9. Kumar V (2002) Introduction to parallel computing, 2nd edn. Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA
10. Macri M, De Rango A, Spataro D, D'Ambrosio D, Spataro W (2015) Efficient lava flows simulations with opencl: A preliminary application for civil defence purposes. Proceedings - 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2015, 328–335. https://doi.org/10.1109/3PGCIC.2015.107
11. Grama AY, Gupta A, Kumar V (1993) Isoefficiency: measuring the scalability of parallel algorithms and architectures. IEEE Parallel Distrib Technol Syst Appl 1(3):12–21
12. Cicirelli F, Forestiero A, Giordano A, Mastroianni C (2018) Parallelization of space-aware applications: modeling and performance analysis. J Netw Comput Appl 122:115–127
13. Was J, Mróz H, Topa P (2016) GPGPU computing for microscopic simulations of crowd dynamics. Comput Inf 34(6):1418–1434
14. Gerakakis I, Gavriilidis P, Dourvas NI, Georgoudas IG, Trunfio GA, Sirakoulis GC (2019) Accelerating fuzzy cellular automata for modeling crowd dynamics. J Comput Sci 32:125–140
15. Giordano A, De Rango A, D'Ambrosio D, Rongo R, Spataro W (2019) Strategies for parallel execution of cellular automata in distributed memory architectures. In: 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 406–413 . IEEE
16. Cicirelli F, Giordano A, Mastroianni C (2021) Analysis of global and local synchronization in parallel computing. IEEE Trans Parallel Distrib Syst 32(5):988–1000. https://doi.org/10.1109/TPDS.2020.3037469
17. Cannataro M, Di Gregorio S, Rongo R, Spataro W, Spezzano G, Talia D (1995) A parallel cellular automata environment on multicomputers for computational science. Parallel Comput 21(5):803–823
18. Willebeek-LeMair MH, Reeves AP (1993) Strategies for dynamic load balancing on highly parallel computers. IEEE Trans Parallel Distrib Syst 4(9):979–993
19. Giordano A, De Rango A, Rongo R, D'Ambrosio D, Spataro W (2020) Dynamic load balancing in parallel execution of cellular automata. IEEE Trans Parallel Distrib Syst 32(2):470–484
20. Giordano A, De Rango A, Spataro D, D'Ambrosio D, Mastroianni C, Folino G, Spataro W (2017) Parallel execution of cellular automata through space partitioning: the landslide simulation sciddicas3-hex case study. In: 2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), pp. 505–510. IEEE
21. De Rango A, Spataro D, Spataro W, D'Ambrosio D (2019) A first multi-GPU/multi-node implementation of the open computing abstraction layer. J Comput Sci 32:115–124. https://doi.org/10.1016/j.jocs.2018.09.012
22. Giordano A, De Rango A, Rongo R, D'Ambrosio D, Spataro W (2020) A dynamic load balancing technique for parallel execution of structured grid models. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 11973 LNCS, 278–290. https://doi.org/10.1007/978-3-030-39081-5_25
23. Avolio M, Di Gregorio S, Mantovani F, Pasuto A, Rongo R, Silvano S, Spataro W (2000) Simulation of the 1992 tessina landslide by a cellular automata model and future hazard scenarios. Int J Appl Earth Observ Geoinf 2(1):41–50