Scalable asynchronous execution of cellular automata

Gianluigi Folino, Andrea Giordano, and Carlo Mastroianni

Citation: AIP Conference Proceedings **1776**, 080006 (2016); doi: 10.1063/1.4965363
View online: http://dx.doi.org/10.1063/1.4965363
View Table of Contents: http://scitation.aip.org/content/aip/proceeding/aipcp/1776?ver=pdfcov
Published by the AIP Publishing

Scalable Asynchronous Execution of Cellular Automata

Gianluigi Folino, Andrea Giordano[a)] and Carlo Mastroianni

ICAR-CNR, via P. Bucci Cubo 7-11b, 87036 Rende (CS), Italy

[a)]Corresponding author: giordano@icar.cnr.it
e-mail: {folino,giordano,mastroianni}@icar.cnr.it

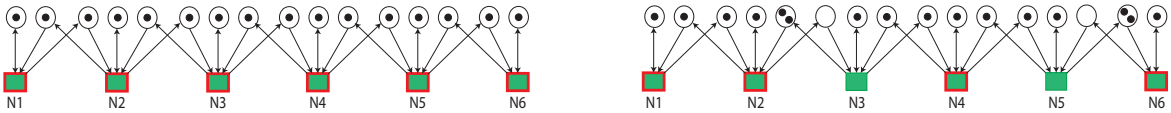
Abstract. The performance and scalability of cellular automata, when executed on parallel/distributed machines, are limited by the necessity of synchronizing all the nodes at each time step, i.e., a node can execute only after the execution of the previous step at all the other nodes. However, these synchronization requirements can be relaxed: a node can execute one step after synchronizing only with the adjacent nodes. In this fashion, different nodes can execute different time steps. This can be a notable advantageous in many novel and increasingly popular applications of cellular automata, such as smart city applications, simulation of natural phenomena, etc., in which the execution times can be different and variable, due to the heterogeneity of machines and/or data and/or executed functions. Indeed, a longer execution time at a node does not slow down the execution at all the other nodes but only at the neighboring nodes. This is particularly advantageous when the nodes that act as bottlenecks vary during the application execution. The goal of the paper is to analyze the benefits that can be achieved with the described asynchronous implementation of cellular automata, when compared to the classical all–to–all synchronization pattern. The performance and scalability have been evaluated through a Petri net model, as this model is very useful to represent the synchronization barrier among nodes. We examined the usual case in which the territory is partitioned into a number of regions, and the computation associated with a region is assigned to a computing node. We considered both the cases of mono-dimensional and two-dimensional partitioning. The results show that the advantage obtained through the asynchronous execution, when compared to the all–to–all synchronous approach is notable, and it can be as large as 90% in terms of speedup.

A PETRI NET FOR THE PARALLEL EXECUTION OF CELLULAR AUTOMATA

In a parallel execution context, a cellular automata can be partitioned by considering a mono-dimensional or a two-dimensional partitioning schema. The assignment of computation to the nodes follows the space partitioning[1][2]: each partition of the territory, or "region", can be assigned to a different computing node, which is in charge of executing the transition rules of all the cells belonging to this specific region. The transition rule of a cell is evaluated on the basis of the states of its neighboring cells. Hence, to execute the transition rules of the cells located in the borders of a region, information must be received from the adjacent computing nodes. For this reason, each node must synchronize with the neighboring nodes.

This parallel execution pattern, including the synchronization, can be expressed by using the Petri net model. Figures 1(a) and 1(b) illustrate the mono-dimensional scenario, for the case that the territory is partitioned into six regions, and the execution is parallelized on six nodes. Six Petri net *transitions*, labeled as N1–N6, are associated with the nodes, and the *firing* of a transition corresponds to the execution of the computation at the corresponding node. Every transition is connected by inbound arcs to three input *places*, and in accordance to Petri net rules [3], the transition is *enabled*, and the computation can start, if all the input places hold at least one *token*. When a transition fires (i.e., the computation is performed at the current step), one token is *consumed* at each input place, and one token is *produced* on each of the output places, i.e., the places connected to the three outbound arcs leaving the transition. One of these output places coincides with the input place of the same transition. The other two output places are input places of the two neighboring nodes: the production of a token in these two places models the transmission of the state of the edge cells of this region to the neighboring nodes and the permission, to such nodes, to execute their computation at the next time step. The two transitions that correspond to the two extreme regions (on the left and on the right) of the partitioned cellular automata are modeled differently in order to consider that these regions have only one neighbor. Indeed, as it can be seen in Fig. 1, only two outbound arcs depart from those transitions.

Numerical Computations: Theory and Algorithms (NUMTA–2016)
AIP Conf. Proc. 1776, 080006-1–080006-4; doi: 10.1063/1.4965363
Published by AIP Publishing. 978-0-7354-1438-9/$30.00



(a) All the nodes are ready to execute.

(b) After execution at N3, N4 and N5. N1, N2, N4 and N6 are ready to execute, while N3 and N5 must wait for the execution at nodes N2 and N6, respectively.

FIGURE 1. Petri net representing the computation at six parallel nodes, in the case of mono-dimensional partitioning.

Figure 1(a) represents the state of the Petri net when all the transitions are enabled, i.e., all the nodes are ready to begin the execution of the current step. The ability to perform the computation is represented by the presence of a red border on the square representing the transition. Figure 1(b) represents the situation after the execution of nodes N3, N4 and N5. N4 is now enabled to execute, because it has performed the previous computation and has received the state of the edge cells from its neighboring nodes N3 and N5. In the Petri net model, this corresponds to the presence of three new tokens at the input places of N4, which means that the synchronization barrier which precedes the next computation at node N4 has been successfully passed. It is also noticed that nodes N3 and N5 are not yet enabled because they are still waiting for the completion of the computation at nodes N2 and N6, respectively. The case of all–to–all synchronization among N nodes, where each node needs to receive information from all the other nodes, is modeled with each transition having N input places, and N outbound arcs connected to all the N nodes.

The Petri net model highlights the advantage of relaxing the synchronization requirements with respect to the classical parallel computation model for cellular automata, in which the synchronization involves all the nodes. When a node needs to synchronize with a limited number of neighboring nodes, different nodes are allowed to execute different time steps. For example, each node can be one step ahead than its direct neighbors, and the gap between the time steps executed by the two nodes located at the two ends of the cellular automata can be as large as N. This is a notable advantage in the case that the computation time varies from node to node and from step to step. The advantage resides in the fact that a longer execution time at one node does not slow down the execution at all the other nodes, but only at the neighboring nodes. As an example, if the nodes located at one end are slower for a period of time, the nodes located at the other end can proceed and execute some additional time steps. In the future, the nodes that are some steps behind can become faster and reach the other nodes, and so on. Overall, this allows the global computation to proceed faster, as will be shown in the next section, devoted to performance results.

The Petri net model can be used also to represent the two-dimensional partitioning schema. Figure 2 contains a portion of the Petri net model, which highlights that the execution at the central node requires the reception of information by the eight adjacent nodes.

RESULTS FOR MONO- AND TWO-DIMENSIONAL SPACE PARTITIONING

A cellular automata is defined through the transition functions executed by the cells that compose the lattice. A transition function executed on a cell uses data coming from the adjacent cells and, at the end of execution, updates the state of the local cell. In general, all the cells execute the same function, even if this is not true in the case of heterogeneous cellular automata. In both cases, however, transition functions of different cells can experience different execution times. Many CA-based distributed frameworks [4] [5] can be used to simulate many complex real world problems such as landslide evolution, lava flows, floods, etc. over distributed architectures. However, with these tools, computation is synchronized at each time-step in an all–to–all fashion, differently from our model that only requires the synchronization among neighboring nodes.

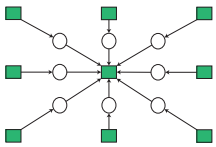


FIGURE 2. Portion of a Petri net representing the computation in the case of two-dimensional partitioning.

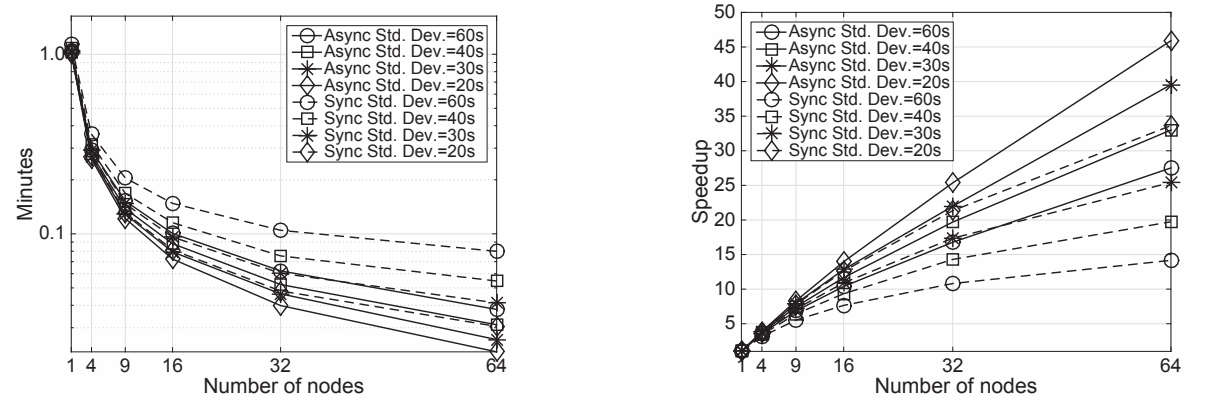
In this section, we consider the case in which all the cells execute the same transition function. Even with this assumption, the execution time varies from node to node and with time: it depends on the substates and on the interaction of the cell itself and can also depend on the workload assigned to the different nodes of the automata. We assume here that the execution time is a random variable with a Gamma distribution. This a quite general assumption since the Gamma distribution, depending on the value of the shape parameter, can represent different types of well-known distributions: e.g., with a shape value equal to one, the Gamma distribution corresponds to an exponential distribution, while with higher values of the shape it has a pdf similar to that of the normal distribution.

The results have been obtained in two ways: by using the well-known Petri net simulator Yasper (http://www.yasper.org/), specifically its "automatic simulation" tool, and through an ad hoc simulator written in Matlab, which reproduces the same behavior of the Petri net model. Results were found statistically identical, with a correlation factor always larger than 0.99. Furthermore, some assumptions were considered in order to simplify the analysis: (i) all the nodes have the same computational power; (ii) the communication overhead needed to communicate the borders' state to the neighboring nodes is negligible w.r.t. the computation time. The latter assumption is realistic in the case that the amount of data needed from neighboring cells is limited. The cases in which these assumptions are not valid are left for future work.

We considered two scenarios for space partitioning: mono-dimensional partitioning and two-dimensional partitioning. In the second scenario, we assume that the number of horizontal and vertical partitions is the same, and therefore the number of regions is a square. The average execution time of a node is obtained by dividing the average serial time – i.e., the average time needed to execute all the automata on a single node – by the number of partitions. The serial execution time is assumed to be one minute. For both the scenarios, we varied the number of nodes and computed two indexes: the average time needed to execute a single step on all the nodes, referred to as T_{step} and the speedup. Furthermore, we compared the all–to–all synchronization approach (each node needs to wait the execution at all the other nodes before advancing at the next step) and the approach discussed here. In the figures, these two approaches are referred to as "Sync" and "Async", respectively.

Figure 3 illustrates the results (T_{step} and speedup) for the mono-dimensional partitionining, when varying the number of nodes. Four different values of the standard deviation of the Gamma distribution were used, from 20 seconds to 60 seconds, in order to consider less or more stable transition functions[a]. The speedup is computed using the well-known formula $S = T_s/T_p$, where T_s is the serial time and T_p is the time necessary to execute the automata on p nodes; it is worth noticing that in the ideal case, the speedup is equal to p, i.e., the number of nodes. It is noticed that the asynchronous approach (continuous lines) performs considerably better than the synchronous approach (dashed lines) for each value of the standard deviation.

As expected, the speedup value is lower when the synchronization delays are higher, i.e., when the variability of execution time is higher. However, the advantage of using the asynchronous approach is more remarkable when

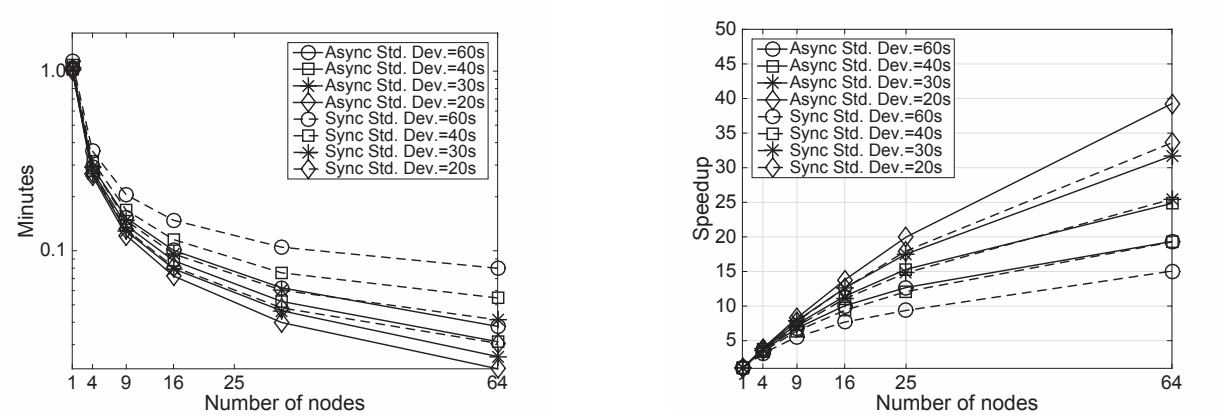


(a) Values of T_{step}.

(b) Values of speedup.

FIGURE 3. Mono-dimensional partitioning: T_{step} and speedup vs. the number of nodes. The automata is equally partitioned among all the nodes. Results are given for the "Synch" and "Asynch" approaches, and for different values of the standard deviation.

[a]The shape value of the Gamma distribution can be obtained as μ^2/σ^2, where μ is the average and σ is the standard deviation.



(a) Values of T_{step}.

(b) Values of speedup.

FIGURE 4. Two-dimensional partitioning: T_{step} and speedup vs. the number of nodes. The automata is equally partitioned among all the nodes. Results are given for the "Synch" and "Asynch" approaches, and for different values of the standard deviation.

the variability is higher. For example, with 64 nodes and standard deviation equal to 20 seconds, the speedup value increases from 33.3 (synchronous approach) to 46.1 (asynchronous approach), thus the improvement is 38.4%. With the same number of nodes and standard deviation equal to 60 seconds, the speedup value increases from 14.3 to 27.3, with an improvement of 90.9%.

Figure 4 shows the values of T_{step} and speedup for the case of the two-dimensional partitioning. Similarly to the mono-dimensional case, the improvement obtained with the asynchronous approach is remarkable, and it increases with the variability of the execution time. We also notice that the speedup decreases with respect to the mono-dimensional partitioning. For example, in the scenario with 64 nodes, standard deviation equal to 20 seconds, and asynchronous approach, the speedup is 39.3 with two-dimensional partitioning, while it is 46.1 with mono-dimensional partitioning. The reason is that with two-dimensional partitioning, a node has to synchronize with eight neighboring nodes instead of only two as in the mono-dimensional case.

CONCLUSIONS AND FUTURE WORK

This is a preliminary study about the advantage that can be obtained when relaxing the all–to–all synchronization approach typically used in parallel cellular automata. We consider the case in which the cellular automata is parallelized with mono-dimensional or two-dimensional partitioning, and each partition only synchronizes with its adjacent partitions before advancing to the next execution step. This asynchronous approach leads to significant benefits, which increase with the variability of the execution times. The advantage in terms of speedup is as high as 90%. In this work, we have assumed that the amount of overhead related to communication is negligible with respect to the computation time, and that the computational load is uniformly distributed over the cells of the automata. Future work intends to investigate what happens when these assumptions do not hold.

REFERENCES

[1] F. Cicirelli, A. Forestiero, A. Giordano, C. Mastroianni, and G. Spezzano, "Parallel execution of space-aware applications in a cloud environment," in *24th Euromicro Int. Conf. on Parallel, Distributed, and Network-Based Processing, PDP 2016, Heraklion, Greece* (2016), pp. 686–693.

[2] F. Cicirelli, A. Forestiero, A. Giordano, and C. Mastroianni, ACM Transactions on Autonomous and Adaptive Systems **11** (2016).

[3] J. L. Peterson, ACM Comput. Surv. **9**, 223–252 (1977).

[4] G. Folino and G. Spezzano, Future Generation Comp. Syst. **23**, 671–679 (2007).

[5] G. Folino, A. Forestiero, G. Papuzzo, and G. Spezzano, Future Generation Comp. Syst. **26**, 87–96 (2010).