Parallel Cellular Automaton Tumor Growth Model

Alberto G. Salguero[1(✉)], Manuel I. Capel[2], and Antonio J. Tomeu[1]

[1] University of Cádiz, Cádiz, Spain
{alberto.salguero,antonio.tomeu}@uca.es
[2] University of Granada, Granada, Spain
manuelcapel@ugr.es

Abstract. "In silico" experimentation allows us to simulate the effect of different therapies by handling model parameters. Although the computational simulation of tumors is currently a well-known technique, it is however possible to contribute to its improvement by parallelizing simulations on computer systems of many and multi-cores. This work presents a proposal to parallelize a tumor growth simulation that is based on cellular automata by partitioning of the data domain and by dynamic load balancing. The initial results of this new approach show that it is possible to successfully accelerate the calculations of a known algorithm for tumor-growth.

Keywords: Cellular automaton · High performance computing
Mathematical oncology · Tumoral growth simulation
Parallel programming · Speedup

1 Introduction

Tumor growth from a transformed cancer-cell into a clinically apparent mass spans a range of spatial and temporal magnitudes. Cellular Automata (CA) can accurately describe the complexity of tumor development [6,20] and this is reproduced by computer simulation. The development of appropriate CA-based software tools will enable tumor prognosis without the need for patients to undergo annoying medical examinations or painful invasive tests.

In order to speed up these computer simulations, recent contributions [16] show advanced techniques for modelling tumor growth such as the use of efficient data structures for supporting deterministic cellular automata (DCA). Multi-paradigm and multiscale models of cancer dynamics [13] have been developed to predict tumor growth and therapeutics.

There have been some approaches based on CA optimization to further extend multiparadigm tumor growth models [2,3,12,14,15,17] and these mainly aim to improve computer simulation performance by guaranteeing efficient data memory program access [16], or by considering the dynamic evolution of the memory space (grids, trees...) that holds crucial data in simulations [18].

In our opinion, the different optimizations based on improving sequential data structure access are not decisive enough to achieve the high-performance computing power actually required by programs simulating cell behaviour. The possibility of using multicore and GPU parallelism as a promising multiplatform and framework to develop new programming techniques to speed-up the simulation computation time has only just started to be explored in recent years [5,8,11].

In order to be able to speed up in parallel programs, this paper presents a CA-based model for tumor growth simulation and identifies the synchronization instructions for implementing this model in a multicore processor in Java.

2 Tumor Growth Model

There are various mathematical definitions of cellular automata. We chose the one established in [1] as the most generally accepted in Computational Sciences, adapted to represent lattice-based biological models [4], and applied to simulate dynamic tumor growth in [16].

One tumor cell in the model is an individual entity that takes up one node of a finite 2D lattice ζ and which can carry out the following actions: migrate to another node on the grid, proliferate by mitosis, die or remain quiescent. A live cell can proliferate by generating a cell's daughter through mitosis whenever there is space available in its neighborhood (given by the Moore neighborhood).

According to Poleszczuk-Enderling's model [16] of reference, the most efficient way of processing tumoral cells consists in keeping a linear list of the tumor cells in the lattice that occupy sequential positions in memory. The list is entirely processed in each simulation step and this yields a new list of cells to be processed in the next one and so on and so forth.

The lattice is used here merely to keep the current state of the tumor. Only the changes in cells are actually written to it. Only two lists are in fact needed to implement the tumoral growth simulation procedure: one listing the cells still to be processed and the second one storing the tumor cells for the next simulation step. This scheme is similar to the one proposed by [10] and which is, to the best of our knowledge, the best sequential solution to the tumor growth simulation problem to date. However, our model uses multicore processors to improve the speed of the tumor growth simulation.

In order to speed up the tumor growth simulation model, two fundamental problems must be solved: firstly, to find a good strategy to maintain balanced cell distribution between threads; and secondly, to prevent access to the data structures (the lattice and lists of active cells) by each thread from blocking other threads for an unacceptable amount of time that could worsen program performance. It is important to note that insertions and deletions on the lattice cell lists must be performed under mutual exclusion to avoid loss of information.

By using different lists for each region it is possible to prevent blocking access to the *current-state* cell list. The *next-state* lists must always be accessed by using blocking access primitives, since cells can migrate between regions when program threads access such lists.

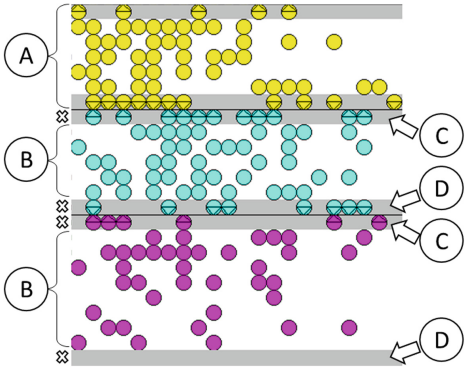

Fig. 1. Lattice division: (A) region assigned to a unique thread; (B) center subregion; (C) bottom part of the seam subregion; (D) top part of the seam subregion

The main objective of the model proposed in this work is to enable concurrent processing of the cells without the need for programming blocking access both to the cell lists and the lattice that holds the current state of the tumor mass. In order to achieve this purpose, the lattice is first divided vertically into as many regions as threads available. Each of these regions is further divided into three parts or subregions: top seam part (C), central part (B) and bottom seam part (D), as shown in Fig. 1. The central parts of the regions are therefore separated by what we call "seams", which are divided into two parts. This layout guarantees the existence of at least two types of subregions between subregions of the same type. This allows the model to concurrently process the cells in subregions of the same type without needing to block access since threads do not enter subregions which are being accessed by other threads. In the case of cell migration to another region, no other thread may access the target subregion at the same time. In order to make use of this property, the proposed model first processes the cells in the bottom seam subregions, then processes those in the central parts and finally those in the top seam subregions (see Fig. 2). All this processing is carried out concurrently and without blocking the threads, except for making them wait until the remaining threads have processed the same type of subregion.

For the model to work efficiently, it is important to perform a correct load balancing between the number of cells that each thread obtains for processing. Once the entire next state of the tumor has been calculated, the main thread is responsible for adjusting the regions, if appropriate.

Subregions must have a minimum height so that no single cell can migrate between non-adjacent subregions. In the proposed model, the minimum height value is δ, the length of the longest displacement of a cell in each simulation step.

2.1 Seams Adjustment During Simulation

As the tumor grows, it is necessary to adjust the seams to equally distribute cells among all the threads. The main thread is responsible for performing this task

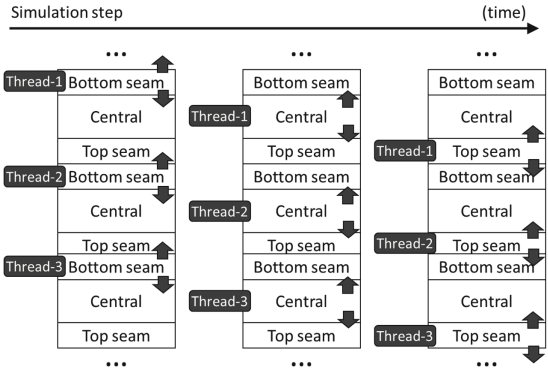

Fig. 2. A step in the parallel simulation

once the remaining threads have finished calculating the next state of the simulation. For reasons of efficiency, the main thread does not update the subregion to which each cell belongs. When the cells are again evaluated by the threads in the next step, the cells are assigned to the corresponding subregion. However, this may cause a cell to directly migrate from the top part of a seam (C) in one region to the central part of another region without passing through the bottom seam subregion, as shown in Fig. 3a. In substep 2, when the upper seam has been moved upward, the a cell still remains in the list of the top seam part (C). In substep 3, both threads simultaneously process the top seam parts (C), and this may cause incorrect concurrent access to the list of cells in the center area (B) or the need to use blocking access primitives. To avoid this situation, the seams are displaced in a three-tier process that is performed in three[1] consecutive and different steps of the simulation, as shown in Fig. 3b:

1. Increase the thickness of the seam in δ rows in the same direction of the displacement (step 2 in Fig. 3b).
2. Decrease and increase the size of the subregions that share the seam in δ rows: the part of the seam that was increased in the previous step is decreased and the other part is increased, maintaining the same seam thickness (step 4 in Fig. 3b).
3. Decrease the thickness of the seam to the default value in the region that has increased its size (step 5 in Fig. 3b).

In this case, cell a can be safely added to list (B) in substep 6 because cell b cannot be processed in the same substep (they are still in different subregions). Since both cells have been assigned to the same thread in substep 3, they cannot be processed at the same instant. For this reason, there is no need to use blocking access primitives in substep 7, when both cells modify the center list (B).

[1] For the sake of simplicity, only substeps where changes have been made to the lists are shown in Fig. 3. Substeps 3 and 4 are actually part of the same overall step.

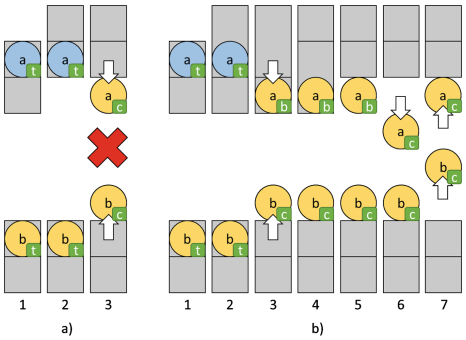

Fig. 3. Incorrect (a) and correct (b) parallel seam adjustment. The small letters in rectangles indicate the current list the cells belong to (t = top seam part, b = bottom seam part, c = center).

The seam adjusting process is automatically started when a significant difference of cells between two adjacent regions is detected. After some tests, it has been observed that the best results are obtained when this difference is greater than 5–15%. The exact value depends on the number of threads in the simulation.

It should also be noted that for reasons of efficiency it is not possible to adjust two consecutive seams. Since the main thread does not reassign migrated cells, it is not possible to determine the number of remaining cells in affected regions. Although this value could easily be calculated, it is important to note that the remaining threads are blocked until the main thread adjusts the seams, so it is crucial for seam adjustment to be performed as quickly as possible.

3 Model Implementations and Measurements

Some experiments have been conducted in order to verify the efficiency of the solution proposed in this work. In these experiments we measured the time taken by a Java application to follow the proposed model and we compared the results with the best known implementation to date [10]. A single cell has been placed at the center of the lattice in every case. The lattice in processed by the threads in parallel, according to model explained in previous section. The application developed follows the Poleszczuk-Enderling model for the simulation of tumoral growth. The model decides whether a cell dies, remains quiescent, or proliferates by mitosis (if there is enough space around the cell to do so) according to a set of probability distributions, which enables us to use a simulation based on a Monte Carlo stochastic submodel. We have employed the default parameters values provided by Poleszczuk-Enderling. More specifically: the cells may be divided ten times before proliferative capacity exhaustion; the probability of division is 1/24; the probability of symmetric division is 0.1; the probability of spontaneous death is 1/24; and the probability of migration is 10/24.

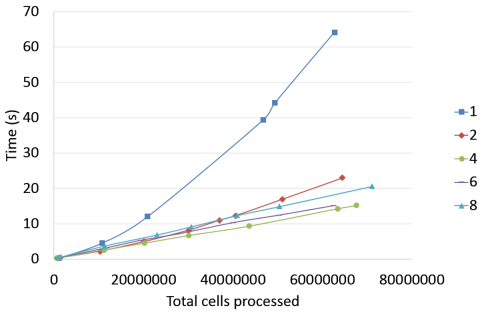

Fig. 4. Execution time for an Intel Core i7 implementation on 4 cores and 8 threads (hyperthreading activated) of the EP-parallel tumor growth simulation

Since the final result in the concurrent version depends on the order in which the cells are evaluated by the threads, it is never possible to obtain the exact same result. The shape of the resultant tumor mass is roughly the same for the same initial configuration and simulation parameters. However, since the border of the tumor is diffuse, the exact number of cells in the tumor mass may vary a lot. Therefore, the time spent on each simulation step cannot be compared among different simulations. Instead, the number of cells processed by time unit is used as reference. Five different configurations with $n = \{1, 2, 4, 6, 8\}$ threads have been used on the i7, while seven different configurations with $n = \{1, 2, 4, 6, 8, 10, 12\}$ threads have been used on the Xeon.

Figure 4 shows that a huge number of cells must be processed in each simulation. Around 300 simulation steps are necessary for the tumor to reach the cell-lattice border regions for the case of $n = 8$ threads, for example. The threads associated with these regions are idle until then. The load will only be equally

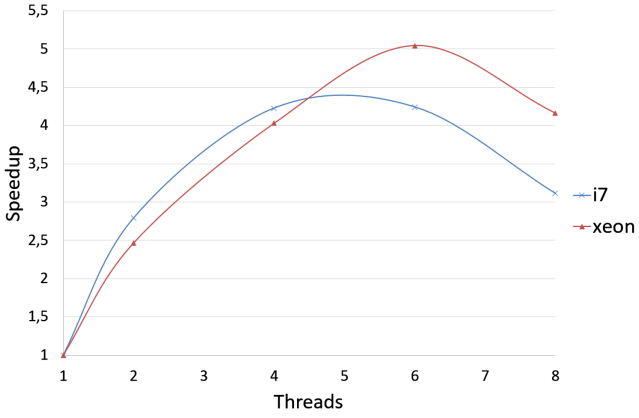

Fig. 5. Speed-up gain for Xeon E5-2670 and Core i7-6700 of EP-parallel tumor growth with respect to the sequential EP simulation

distributed among all the threads after 500–600 simulation steps. The speed-up gain for $\{2, 4, 6, 8\}$-thread configurations is shown in Fig. 5. The same behavior is also observed when the experiment is performed in one of the nodes of the University of Cádiz. No noticeable improvement has been found for more than six threads.

4 Conclusion and Future Work

The application of parallelization techniques to cellular automata-based tumor growth simulations on multicore and many core or processor clusters is a research field which has not yet reached full maturity. By introducing a data parallel scheme with several computing threads, we have proposed a parallelization approach for a basic tumor growth simulation model. Different processing loads were deployed on selected multicore processor architectures for a Java parallel program and compared with the tumor growth model with dynamically growing data domains given in reference [16]. This model manages to accelerate the simulation algorithm by optimizing the cellular automaton data structures while allowing data regions to be accessed by single thread at a time.

The obtained results showed better speedup in computations than the tumor growth simulation program of reference in [16]. However, there is still room for the improvement of the model. Good acceleration is obtained when using half the number of cores in the architecture. From then on a worsening on the performance is observed. In the tests we have carried out, it has been detected that the load-balancing mechanism is not fast enough to equally distribute the workload among all the threads when the tumoral mass grows. This is because the edges of the tumor are very diffuse, which causes very rapid changes in the number of cells in each region of the lattice.

Our future work will be directed towards obtaining a new implementation of our tumor growth model for GPU based on the data domain model array with dynamic growth, initially based on the sequential model in [16]. The proposed model is being improved with the inclusion of dynamic tumor growth characteristics that have been mentioned by various authors [13], such as tumor vascular prominence (angiogenesis) or tumor nutrient intake, which can be modeled using hybrid lattice-gas cellular automata [9].

References

1. Adamaztky, A., De Lacy, B., Tetsuya, A.: Reaction-Diffusion Computers. Elsevier (2005)
2. Alarcón, T., Byrne, H.M., Maini, P.K.: A cellular automaton model for tumor growth in inhomogeneous environments. J. Theor. Biol. **225**(2), 257–274 (2003)
3. Aubert, M., Badoual, M., Fereol, S., Christov, C., Grammaticos, B.: A cellular automaton model for the migration of glioma cells **3**(2), 93–100 (2006)
4. Bandman, O.: Implementation of large-scale cellular automata models on multicore computers and clusters. In: International Conference on High Performance and Simulation (HPCS), 1–5 July 2013. https://doi.org/10.1109/HPCSim.2013.6641431

5. Blecic, I., Cecchini, A., Trunfio, G.A.: Cellular automata simulation of urban dynamics through GPGPU. J. Supercomputing **65**, 614–629 (2013). https://doi.org/10.1007/s11227-013-0913-z
6. Capel-Tuñon, M.I., et al.: Towards modal modelling of biological systems. Technical report: Michigan State University, pp. 1–12 (2008)
7. Chopard, B., Droz, M.: Cellular Automata in Modeling of Physical Systems. Cambridge University Press, Cambridge (1998)
8. D'ambrosio, D., Filippone, G., Rongo, R., Spataro, W., Trunfio, G.A.: Cellular automata and GPGPU: an application to lava flow modeling. Int. J. Grid High Perform. Comput. (IJGHPC) **4**(3), 18 (2012)
9. Deutsch, A., Dorman, S.: Cellular Automata Model of Biological Patterns. Characterization, Applications and Analysis. Birkhuser (2005)
10. Enderling, H., Anderson, A., Chaplain, M., Beheshti, A., Hlatky, L., Hahnfeldt, P.: Paradoxical dependencies of tumor dormancy and progression on basic cell kinetics. Cancer Res. **69**, 8814–8821 (2009)
11. Gibson, M.J., Keedwell, E.C., Savic, D.A.: An investigation of the efficient implementation of cellular automata on multi-core CPU and GPU hardware. J. Parallel Distrib. Comput. **77**, 1125 (2015)
12. Jiao, Y., Torquato, S.: Emergent behaviors from a cellular automaton model for invasive tumor growth in heterogeneous microenvironments. PLOS Comput. Biol. **7**, Article ID: e1002314. https://doi.org/10.1371/journal.pcbi.1002314
13. Khan, M.A., Shefeeq, T., Kumar, A.: Mathematical modeling and computer simulation in cancer dynamics. Int. J. Math. Model. Simul. Appl. **4**(3), 239–254 (2011)
14. Patel, A.A., Gawlinski, E.T., Lemieux, S.K., Gatenby, R.A.: A cellular automaton model of early tumor growth and invasion: the effects of native tissue vascularity and increased anaerobic tumor metabolism. J. Theor. Biol. **213**(3), 315–331 (2001)
15. Piotrowska, M.J., Angus, S.D.: A quantitative cellular automaton model of in vitro multicellular spheroid tumour growth. J. Theor. Biol. **258**(2), 165–178 (2009)
16. Polesczuk, J., Enderling, H.: A high-performance cellular automaton model of tumor growth with dynamically growing domains. Appl. Math. **5**, 144–152 (2014)
17. Ribba, B., Alarcón, T., Marron, K., Maini, K., Agur, Z.: The use of hybrid cellular automaton models for improving cancer therapy, pp. 444–453 (2004)
18. Rybacki, S., Himmelspach, J., Uhrmacher, A.: Experiments with Single Core, Multi Core, and GPU-based computation of cellular automata. In: 2009 First International Conference on Advances in System Simulation, pp. 62–69 (2009)
19. Tomeu, A.J., Salguero, A.G., Capel, M.I.: A parallelisation tale of two languages. Ann. Multicore GPU Program. **2**(1), 81–94 (2015)
20. Trisilowati, Mallet, D.G.: Experimental modeling of cancer treatment. ISRN Oncology, **2012**, Article ID 828701 (2012). https://doi.org/10.5402/2012/828701