Ecological Modelling with Cellular Automaton Software Implemented in GPU System

Éva Hajnal and Tamás Bajzát

Óbuda University, Alba Regia University Center
H-8000 Hungary Székesfehérvár Budai út 45, Hungary, hajnal.eva@arek.uni-obuda.hu bajzat.tamas@gmail.com

Abstract: *A modelling software, based on the cell automaton principle, was developed to help the investigation of the abundance distribution, and the change of species number in freshwater phytoplankton communities. The task is computationally intensive, but due to the data parallel computing model, it is conveniently implemented in PC by the up-to-date GPU cards. In the development we have used a re-thought, and upgraded successor of the Nvidia G80 architecture, Fermi-GF104 architecture, and the associated CUDA programming environment in C/C++ language environment. The performance of the modelling software was tested and the CGMA (compute to global memory access) ratio was determined. It was ascertained that few magnitude speedup can be reached using the GPU systems supplemented the traditional serial data processing way of CPU-s.*

Keywords: *cell automaton, GPU, massive parallel algoriths, ecological model*

1. Introduction

The behaviour of ecosystems is deeply examined for centuries, but only few studies have been published about communities in which the number of species is large, but from the aspect of interspecific interactions seem to be quasi-neutral. Such community is the phytoplankton, with quantity and species composition it reflects the state of the freshwater and the see, and it is one of the main participants in the oxygen production of the Earth. Our aim is to supplement our results in database analysis of the phytoplankton of different lakes and our mathematical findings, and developing software for modelling the abundance distribution and the species number changes of it. We decided to implement our software on the base of the cellular automaton modelling principle.

Cellular automaton (later CA) models have been used for nearly 25 years, and their theoretical bases are thoroughly clarified [1, 3], but till now they seem to be promising modelling methods. These models work on the level of entities, and try to create complex phenomena. In this model the state of cells and their behaviour depends on the local or wider neighbourhood and from their own previous state. This model generally makes possible the demonstration and visualization of the formation of temporal and spatial patterns but only in few cases facilitates quantitative approaches [3, 4]. Their application could be the first step in scientific understanding, which with calculations by simply rules creates phenomena, thus preceding the rigorous mathematical and physical models [3].

Against the previously mentioned advantages, there are some disadvantages of the method, namely it is a computationally intensive task with a large number of operations caused by the large number of data elements and the large number of iterations. Making a compromise, the traditional CA models used quite small cell lattice, two-state cells, and the cell state transitions were calculated with bitwise operations. Contrary of its simplicity, these CA models were effective to demonstrate a wide range of phenomena, such as angiogenesis [3], urbanization [2], heat conduction in fluids, signal processing in the heart [3] etc. The importance of the method inspired the development of a cell automaton parallel computer with a multiprocessor, in which the cell transition rules were programmable [1]. This computer worked with floating point operations in the cell lattice.

The modern GPU systems, with affordable cost can execute high performance scientific computing on personal computers, and the good price per performance ratio makes possible the researchers with restricted resources to use them [4]. The application of GPU-s encouraged software engineers to revive CA models in a modern architecture with enlarged functionality. The enlargement of the cell lattice and the number of iterations, the possibility of computing with complicated cell transition rules are guaranteed by this technical background. Moreover, the floating point operations became faster, and in the transition rules we can take into account not only the local neighbours, but distant cells with proportions of the distance, furthermore cells can move in the cell space.

Our aim is to implement a model, where the number of species (n>100) is large, the abundance of the species can differ in few magnitude, but the interaction between the species is restricted to the competition for the nutrients. [8] In this model each species has special own reproductive, death and settling rate and these rates are changing stochastically in time [6].

Today's graphics cards in computing performance manifoldingly prevail over CPUs. Since 2007, from the appearance of the Nvidia G80 type chip, they can be programmable to general computing tasks [5, 10, 11]. This was the first card, which worked on newly developed general processing units instead of the special vertex and pixel shader. These new processors can be interpreted as simply, scalar ALU-s which execute the same simple instructions parallel on each data of the input stream thus creating the output stream. This single instruction multiple thread (SIMT) architecture results in giant computing performance.

Until the CUDA platform was not released, the programmers had to write shader programs to use the graphics card's performance for their own general algorithms. A shader program just could work with special data structures, and it had a special programming technique, because a conventional shader program is used in the graphics pipeline. However, it had a great parallel performance, although the programming was complex and problematic.

In 2007 the Nvidia released the CUDA C platform, and it changed our software developing aspect. It was provided for programming the CUDA capable GPUs in C programming language with only few constraints. These constraints derive from the architecture, and computing model differences between the CPUs and GPUs.

Furthermore, among others, the Compute Unified Device Architecture (CUDA) makes possible programming this device on high level programming languages such as C or C++. This work only needs the knowledge of parallel programming paradigms [7,9], and the advantage is the instruction execution on different, "C" type data structures. The main functional parts of a CUDA program help co working of the CPU and GPU. The first step is the memory allocation on the graphics device, which is followed by the data load onto the allocated memory blocks. After this preparation the data processing on the GPU begins. When the GPU code block is finished, the data have to be loaded onto the system memory, for further process on CPU. According to this program structure [10, 11], the available instructions could be grouped into three based on processing and calling. The host code is executed on the CPU. This is a clear C/C++ code without any constraints. Its responsibility is the arrangement of the GPU. The kernel code is called by the CPU but executed on the GPU, and the device code is callable from the kernel code, and its functions are executed on the GPU.

2. Algorithm and Implementation

In the CUDA programming model the graphics card plays the role of a coprocessor, which helps data parallel processing, and in this work it is supported by its own DRAM memory. Processing performance may be enhanced considerably with CA algorithms in which the data matrices are used in data storage and their simply cell transition rules, exactly match the computing and hardware possibilities. Its performance may be further enhanced by the skilled usage of GPU cards three or four level special memory model.

2.1. Rules of the CA Model

The rules are executed in three phases (Fig. 1). Their first and second stages deal with empty cells and the third works on the "living" cells. The first phase simulates the settling of different species. Each species has an own settling possibility value (P_s), and if a randomly generated possibility value of a randomly selected species remain below this P_s, an individual of this species can settle into an empty place of the cell lattice (Fig 1/B). The second phase simulates the reproduction with generating a child into an empty cell. This reproduction occurs with random possibility according to the reproductive rates of each species by the random order of the neighbourhood. Each species has two reproductive rates, first determines the base reproduction (P_{r1}, P_{r2}) and the second determines the reproduction under favourable conditions (Fig 1/C). The length of the favourable period is also a species specific feature. The third phase deals with the age and death. Each cell's age is increasing in every cell lattice cycle, and the cell dies, when it reaches the species specific lifetime (k).

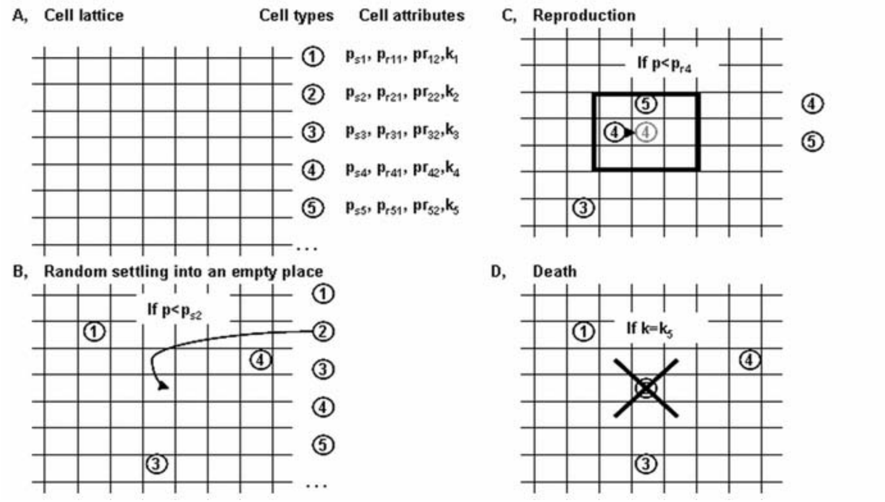
A, Cell lattice
Cell types
Cell attributes
① $p_{s1}, p_{r11}, pr_{12}, k_1$
② $p_{s2}, p_{r21}, pr_{22}, k_2$
③ $p_{s3}, p_{r31}, pr_{32}, k_3$
④ $p_{s4}, p_{r41}, pr_{42}, k_4$
⑤ $p_{s5}, p_{r51}, pr_{52}, k_5$
...
B, Random settling into an empty place
If $p<p_{s2}$
C, Reproduction
If $p<p_{r4}$
D, Death
If $k=k_5$

Figure 1: The rules of the CA modelling software. A, The cell lattice may contain large number of cell types, each of them have an ordinal number, and probability attributes for settling, reproduction and death. B. A random settling into an empty place in the cell lattice. C, The cells may reproduce themselves into a neighbouring empty place. D. The cells die when their lifetimes reach the number of reproductive cycle targeted by the k value.

2.2. The Structure of the CA Software

Our CA modelling software works on a cell lattice, which logical representation is a two dimensional data matrix. According to the data processing ability of the device it is one dimensional data vector. This lattice is loaded with cells by different distributional patterns, and the simulation calculates the next states by cell state transition rules with the required number of iterations. Random samples are taken out during the simulation in the required steps of iteration. The summarized results of the samples are exported into MS-SQL database. The GPU module is responsible for the simulations.

A cell of the lattice, which is targeted with the modelling software defined as a structure, because this type is also favourable by the ANSI C, consequently, it is available in CUDA C functions without any restrictions. The advantage of this type is the same and common variable types in GPU and CPU code. The most important data of the structure are the species identity code, the actual and the maximal age of the cell, the two types of reproductive rate, settling rate, neighbours indices, number of neighbours, and a pointer to this data structure if the cell is alive.

Cell and Cell Factory Class

The cell space class contains the distribution and rules objects and the one dimensional array for the cell storage. At the beginning this data block is uploaded with cells by the distribution class, and the simulation is executed by the rule object with the required number of iterations.

The cell factory class contains the species descriptions, imports them from a file, and creates cells by the requests. This object is able to export the data of the cells, and its functions are used also by the statistical module.

Rules Class

The rules of the software are inherited from a common basic rule class. It has functions for the execution of cell transition rules in the existence of GPU and in absence of it, on CPU. These functions are called by the cell lattice object. By the redefinition of this basic class, whatever rules can be applicable. This algorithm formally calculates the same operations on all cells, by this means reduces the program control instructions, and makes possible the threads to execute the same instructions.

Distributions Class

The investigations of the distribution patterns of real communities showed, that it is not exact enough to approximate the distribution with a single classical distribution function. This CA model uses the resultant of two types of distribution functions (broken logarithmic, lognormal or smooth distributions). The different distributions are also inherited from a common basic class, and the program can be enhanced with new types of abundance distributions. This object calculates the number of each species according to the available distribution function and these cells are settled into the cell lattice into random positions by the cell factory. In the performance tests only one logarithmic distribution was used with a changeable K ratio of a geometric series.

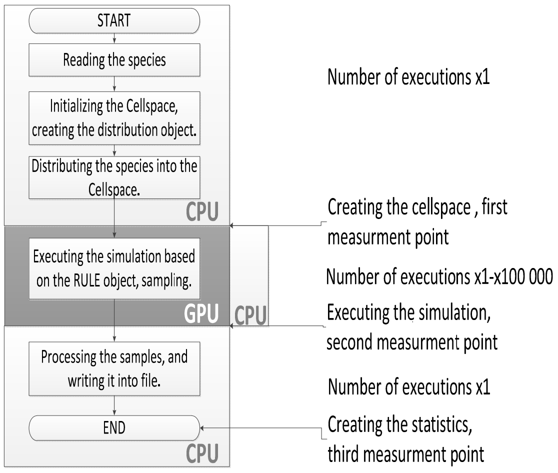
START
Reading the species
Initializing the Cellspace, creating the distribution object.
Distributing the species into the Cellspace.
CPU
Executing the simulation based on the RULE object, sampling.
GPU
CPU
Processing the samples, and writing it into file.
END
CPU
Number of executions x1
Creating the cellspace , first measurment point
Number of executions x1-x100 000
Executing the simulation, second measurment point
Number of executions x1
Creating the statistics, third measurment point

Figure 2: The modules of the CA software and the measurement points for the performance testing

Statistics Class

Sampling is accomplished uniformly in the required iteration steps. It means the random generation of the required number of indices from the cell lattice, histogram is calculated and data are exported into an MS-SQL database. Advanced statistical calculations can be executed with special software after all.

2.3. Performance Measurement

For the performance testing we settled two inner time measurement points into the software (Fig. 2). The first inner point is at the beginning of the GPU module, and the second is at the end of that module. So we can measure the processing time of the three basic module of the program, the input and initialization block, the simulation block, and the statistical block. For rigorous time measurements the Windows API functions were used.

3. Results

The aim of the first measurements was to verify, which program block determinates mainly the performance of the program execution (Fig. 3). The simulation module is the main block from the aspect of time usage, and the execution time of this block depends basically on the program parameters.

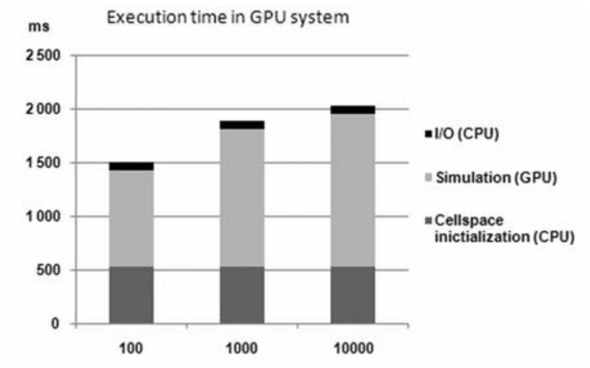

Figure 3: Performance measurements on changing the number of iterations in GPU(execution times in milliseconds, number of species=10, size of cell lattice=10 000) and the execution time dependence from parameters in each block

The second measurement (Fig. 4) is the examination of the execution time dependence on the size of the cell lattice. In this attempt the number of species (10) and the number of iterations (1000) are constant and the cell lattice's size is 32x32, 64x64, 256x256, and the experience is similar. The difference in speed is caused by the GPU module, which is the main part of the program from the aspect of the total execution time. The trend of execution time changing rises.

The summarized efficiency comparison on Figure 5 shows the relative advantage of the parallel GPU execution to the serial CPU execution.

Number of species:	10							
Number of iterations:	1000							
Number of cells:	1024		4096		16384		65536	
	CUDA	CPU	CUDA	CPU	CUDA	CPU	CUDA	CPU
Cellspace creation(ms):	47	47	187	187	764	749	3026	2995
Executing of the simulation(ms):	1092	3885	1154	7613	1529	19454	1820	198262
Writing the result into a file(ms):	78	78	78	78	78	78	78	78
Total(ms):	1217	4010	1419	7878	2371	20281	4924	201335

Figure 4: Performance measurements on changing the size of the cell lattice (execution times in milliseconds) comparing a GPU system with a CPU system

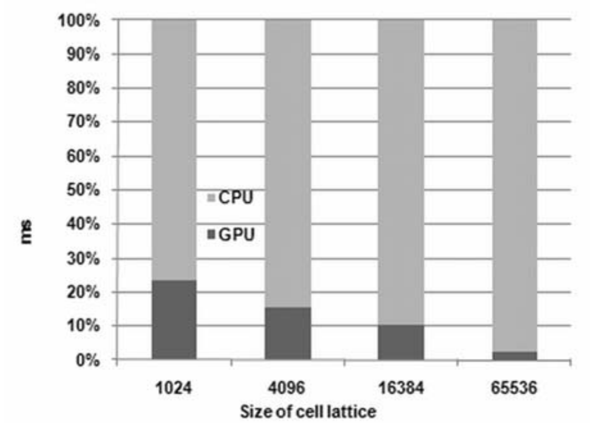

Figure 5: Summarized relative execution time comparison of the software between CPU and GPU systems. (Species number=10, number of iteration=10000)

The calculated Compute to Global Memory Access (CGMA) ratio is a good indicator of the parallel algorithm's speedup. In the GPU module it decreased almost until 0, because in the software, the access to global memory is restricted to the data export, if the local memory of the GPU card is full, and to the export of statistical data. The instructions in GPU simulation module can operate mostly with the device own memory types.

Conclusions

A CA model was developed for the investigation of a real ecological system, which is in our focus [6, 8], and the software performance was tested [12]. The results (Fig. 5) show that serial data process by CPU is commensurable with parallel process of the GPU at small number of iteration due to the enhanced instruction set, and caching. But with the enlargement of the computing task, the GPU prevails the CPU with some orders of magnitude. Because the 10 fold enlargement of iterations is a real 10 fold enlargement of the instruction number from the aspect of the GPU, but 10*n (n is the number of the cells, here ~10000) from the aspect of the CPU. On the testing personal computer the lowest computational task does not cause trouble to CPU systems, but at larger cell lattice and large number of iterations there is some order of magnitude advantage in the relative speed of the parallel algorithm. This program is expected to execute for modelling large multitude, and large number of iterations, near the upper level of the testing dataset. Additionally this program will be executed on a large data series in batch mode, so the performance of the GPU is really need.

The key of the accessed performance is the software planning, namely the module structure of the software and the implementation of the modules on the appropriate device; the high computationally intensive block was implemented on the GPU but the high I/O intensive block was implemented on the CPU. The local algorithm considerations caused further enhancement of the performance of the parallel algorithms.

The software needs further development. The statistical block needs some development for indicating the stability of number of individuals, species in the cell lattice, with statistical evaluation of the increasing or decreasing living cell and species number and the system's entropy.

In summary the low expense parallel programmable devices changes our point of view. Software engineering for personal computers emphasizes solutions which were applied on large systems and may change our model choosing decisions, and facilitates the financially weakly supported theoretical ecological research.

Acknowledgment
We thank Mariann Machata and Balázs Almádi for helpful hints.

References

[1] T. Toffoli, and N. Margolus, Cellular automata machines: a new environment for modeling. Cambridge, MA: MIT Press. 1987

[2] M. Batty, and Y. Xie , "From cells to cities" Environment and Planning B: Planning and Design 21 Supplement, s31 – s48 1994

[3] G. B. Ermentrout and L. Edelstein-Keshet, Cellular automata approaches to biological modeling, Journal of Theoretical Biology Vol. 160, pp. 97-133, 1993

[4] J. Tran, D. Jordan and D. Luebke, New challenges for cellular automata simulation on the GPU - SIGGRAPH, Los Angeles. ACM. Poster, 2004 – Citeseer

[5] D. B. Kirk, W.W. Hwu, Programming Massively Parallel Processors – Elsevier,Burlington, 2010

[6] Padisák J, Hajnal É, Krienitz L, Lakner J, Üveges V, Rarity, ecological memory, rate of floral change in phytoplankton – and the mystery of the Red Cock. HYDROBIOLOGIA 653: pp. 45-64. 2010

[7] R. Nagy, and M. Seebauer, Parallel Computing in the Electrical Engineering Curriculum, ITHET 2002 3rd International Conference on Information Technology Based Higher Education and Training, 2002

[8] J. Lakner, É. Hajnal., G. Lakner and J. Padisák, Statistical mathematical modelling for multitude number estimation of rare and frequent species. Ecological Modelling under submission

[9] M. Sarnovský, P. Butka and J. Paralič, "Grid-based Support for Different Text Mining Tasks" Acta Polytechnica Hungarica Journal of Applied Sciences, vol. 6, Issue Number 4, pp. 5-25, 2009

[10] NVIDIA CUDA C Programming Guide,http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf , 20.10.2011.

[11] CUDA API REFERENCE MANUAL, http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_Toolkit_Reference_Manual.pdf, 20.10.2011

[12] Bajzat, T.; Hajnal, E., "Cell Automaton Modelling Algorithms: Implementation and Testing in GPU Systems," *Intelligent Engineering Systems (INES), 2011 15th IEEE International Conference on* , vol., no., pp.177-181, 23-25 June 2011