Conway's Game of Life accelerated with OpenCL

Thomas Rumpf

Friedrich-Schiller-University Jena, Germany
Department of Mathematics and Computer Science
`thomas.rumpf@uni-jena.de`

Abstract. We introduce a massively data-parallel implementation of John Conway's cellular automaton Game of Life in-silico which provides the user with a high flexibility to modify initial settings and progression rules. Written in C/C++ and employing the open programming interface OpenCL our program utilizes the parallel computing architecture of modern graphics cards leading to a significant speed-up in estimating artificial life-form generations in comparison to ordinary CPU-based implementations. Custom input files for specification of initial configurations as well as a variety of features to control the course of the game on the fly contribute to improve the productivity in conducting experimental studies using our cellular automaton framework.

Key words: Game of Life, OpenCL, OpenGL, parallel computing, GPGPU

1 Introduction

John Conway designed the cellular automaton Game of Life in 1970 to prove that it is possible to build a simple yet powerful machine which can simulate the basic concepts of life. The outcome of his automaton had to be unpredictable meeting three desiderata: firstly there should be no initial pattern for which there is a simple proof that the population can grow without limit, secondly initial patterns that apparently do grow without limit and thirdly initial patterns fading away, settling into a stable configuration or entering an oscillating phase [2].

He achieved these goals employing a simple two-dimensional board of equally shaped cells which had only two states – dead or alive. Starting with a self-chosen allocation of alive and dead cells the game evolves with each generation while there are some simple rules which have to be applied simultaneously at each cell on the board after each generation. If a dead cell has exactly three alive cells in its Moore neighborhood it is reborn in the next generation. This type of neighborhood comprises the eight cells surrounding a central cell on a two-dimensional square lattice. Alive cells survive the current generation if they have exactly two or three alive cells as neighbors. In all other cases alive cells die and dead cells stay dead. These rules can be symbolized as B3/S23 which are two lists of numbers representing the birth and survival requirements of dead and alive cells.

Our implementation of the Game of Life evolved as a project for Programming with CUDA and Parallel Algorithms at the University of Jena. Here we aimed to identify a problem which is usually solved with CPUs but could be improved by the use of GPUs which are specialized for compute-intensive, highly parallel computation. With the introduction of the Compute Unified Device Architecture (CUDA) by NVIDIA in 2006 programmers are now able to use these computing power for general purposes (GPGPU) [3] [4].

Forward-looking we decided to use OpenCL for this project which is an open royalty-free standard for general purpose parallel programming across CPUs, GPUs and other processors [6]. This means the user is free to use GPUs from NVIDIA [3] or ATI [5] and other parallel processors found in servers and handheld or embedded devices.

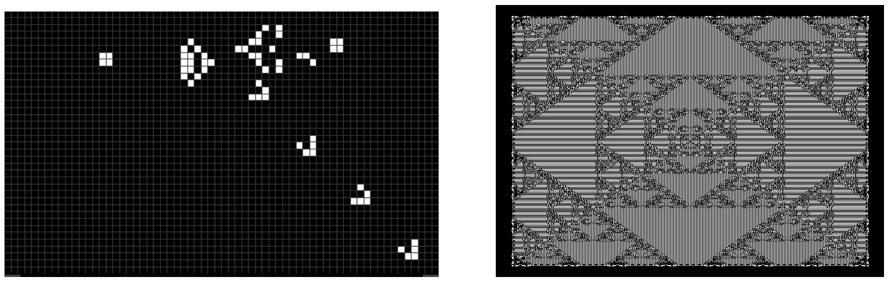
Fig. 1. Bill Gosper's glider gun using rule B3/S23 and generation 235 of a pattern similar to the sierpinski triangle starting with a single alive cell and using rule B1/S12

2 Challenges

One of the main challenges of implementing the Game of Life with C/C++ and OpenCL was the potential infinite board of cells. Since memory boundaries are finite on both the host system and the GPU, we decided to use arrays with a fixed size representing the current and next generation but also implemented a switch for wrapping and clamping mode of the board. For example in wrapping mode a cell in the middle of the left border reads the state of the five surrounding cells in its local environment and reads the state of the three cells which are in the middle of the right border. In clamping mode all requests for states of cells outside of the fixed board return the value of a dead cell.

The OpenGL extension of C/C++ is used for visualizing the board and its cells. When we were developing this program with the latest NVIDIA driver version 195.17 it was not possible to display board generations that were calculated and stored on the GPU directly with the GPU. Our implementation needs to copy each generation back to the host system and then use OpenGL for a graphical output, again on the GPU. This slows down the calculation of more generations considerably. Thus, we implemented a switch that allows skipping the visualization of certain generations by calculating more generations on the

GPU while the host system copies a single generation from the GPU. Further speed-ups can be achieved by implementing the interoperation between OpenCL and OpenGL which will be included in future drivers of NVIDIA and ATI.

3 Optimizations

From a technical point of view we applied certain optimizations that speed up the process of calculating generations. Instead of using simple arrays OpenCL offers two-dimensional image objects which are usually used as memory for textures and cannot be directly accessed using a pointer but with built-in read and write functions. Each image element is a four-component vector which represents the values red, green, blue and the opaqueness of the color as an alpha value. In our case a dead cell has the color black (R0,G0,B0,A1) and a alive cell is white (R255,G255,B255,A1). The advantage of images is that the elements of a read function are cached which offers a higher performance when reading image locations that are close to each other. Additionally images offer the option to easily toggle between wrapping and clamping mode. The OpenCL standard specifies that an image object has a maximal width and height of at least 8192 pixel so the latest GPUs may support even larger images. Unfortunately the most recent ATI driver version 10.2 offered no image support thus right now our program is limited to the use of NVIDIA GPUs.

Another relevant optimization is focused on applying the rules for each generation. To determine the next state of a cell only two values are required – its current state and the number of alive cells in its Moore-neighborhood. To avoid lots of nested if-then constructs to simulate a rule like B3/S23, we have used a look-up table where each entry represents the state of a cell in the next generation if it has the amount of alive neighbors according to the applied rule.

With these optimizations the program is approximately 16 times faster than a naive CPU version of the Game of Life. Compared to fastest known algorithm HashLife by Bill Gosper [7] which uses forward calculation our program is at best only five times slower than the CPU implementation of HashLife which is included in the application Golly [8]. However, to implement this algorithm with OpenCL can be seen as a challenging task due to the memory limitations on a GPU. All these tests were made on a machine with an Intel Xeon E5420 @ 2.50GHz, an NVIDIA GTX 260 and different board sizes ranging from 256x256 to 2048x2048. Graphics cards of future generations would give even better results.

4 Features and limitations

Besides being able to start with a random assignment of alive and dead cells our program can also read patterns for starting populations from human-editable RLE (Run-length encoding) files [9]. If a rule is specified in the file it will be used instead of the default B3/S23. This default rule can also be changed to any rule that can be described in the same way, for example B1357/S02468 or B1/S12. Here is an example of a command to start our program with a random population whose density of alive cells is 40%, follows the rule B1/S12 and covers a board with a width and height of 1024 cells: `./GameOfLife -r 0.4 -l 12/1 1024`.

The OpenGL interface offers zooming in and out of the board or moving around giving comfortable access to the displayed generations. One can also enable a grid for the board however, with very large image sizes this can slow down the display speed. On the other side if the display of generations is too fast one can adjust the waiting time between generations or even stop the game after each generation.

There is one limitation of the program which regards the values of the board size. When using the default wrapping mode images sizes are constrained to be a power of two and the width and height needs to be the same value, too. The problem is related to inaccurate values when reading the state of a cell from the image object as normalizing the image coordinates in wrapping mode to a fixed range is mandatory in OpenCL.

5 Outlook

With its easy to use interface for dynamic rules and starting patterns this program offers researchers on cellular automata a fast and efficient way to explore new interesting patterns [10] [11].

Acknowledgment

I would like to thank Jens K. Mueller and Waqar Saleem for sharing their knowledge of CUDA and providing helpful suggestions for this Game of Life implementation and Thomas Hinze and Sascha Schaeuble for advice.

References

1. FSU Jena, Department of Mathematics and Computer Science: Programming with CUDA, `http://theinf2.informatik.uni-jena.de/For_Students-p-9/Lectures/Programming_with_CUDA-p-41/WS_2009_2010.html#projects`
2. Martin Gardner – Mathematical Games (October 1970), `http://www.ibiblio.org/lifepatterns/october1970.html`
3. NVIDIA CUDA, `http://developer.nvidia.com/object/gpucomputing.html`
4. Cecilia, J. M., Garcia, J. M., Guerrero, G. D., Martinez-del-Amor, M. A., Perez-Hurtado, I., Perez-Jimenez, M. J.: Simulation of P systems with active membranes on CUDA. Briefings in Bioinformatics (December 2009)
5. ATI Stream Technology, `http://www.amd.com/stream`
6. Khronos OpenCL Specification, `https://www.khronos.org/registry/cl/`
7. Gosper, R. Wm.: Exploiting regularities in large cellular spaces. Physica D: Nonlinear Phenomena, Volume 10, Issue 1-2, p. 75–80 (1984)
8. Golly: an open source, cross-platform appl., `http://golly.sourceforge.net/`
9. Cellular Automata file formats: RLE, `http://psoup.math.wisc.edu/mcell/ca_files_formats.html#RLE`
10. Game of Life News, `http://pentadecathlon.com/lifeNews/index.php`
11. LifeWiki, `http://conwaylife.com/wiki/`