Use NVIDIA CUDA technology to create genetic algorithms with extensive population

Mgr inż. Maciej Kołomycki

Supervisor: Prof. dr hab. inż. Leszek Wojnar

Abstract

This article presents a method of implementation genetic algorithm in CUDA. Used algorithm operat on a large population and a complex genotype, so that it exceeded the size of the cache memory. It is not completely transferred to the graphics card. It consists of modules that run on the CPU and are synchronized through it. Calculations were based on weak, but widely available graphics cards to test the ability of acceleration algorithms at low cost.

Keywords

CUDA, genetic algorithms, parallel processing on GPU

1. Introduction

In my work I tested the possibility of using CUDA technology as a tool for accelerating the calculation of genetic algorithms with large populations and complex genotypes. In many publications sought to test the possibility of using graphics cards to accelerate the calculation of genetic algorithms. They amounted to accelerate the evaluation of individuals element of the population [1] or processing of the entire genetic algorithm on the GPU [2]. Before the rise of CUDA technology, the use of graphics cards to calculate these algorithms was relatively difficult [3-5]. Elements had to be present in the form of textures. Genetic operations were complicated in the implementation because of unnatural data record. The situation changed after the technology CUDA. It has allowed direct access to all cores and easy to use graphics cards to advanced numerical calculations. There are many publications on the implementation of genetic algorithms In CUDA technology [6-9]. The best results were achieved in a multi-population algorithms because of possibility to use blocks as "islands" for each population and shared memory to store these populations. This approach eliminates the problem of the lack of synchronization between the blocks. However, it brings some inconvenience. Shared memory is relatively small (16 KB in the GT200 architecture), which place restrictions on the size of the population, or forced to put it in the global memory which is much slower.

In my work I have tried to concentrate on the possibilities of use the graphics card to speed up the calculations one-population genetic algorithm with a large population (greater than the size of the shared memory). I conducted calculations on the GT540M to test the capabilities of cheaper mobile graphics accelerators for use in numeric applications.

2. Genetic algorithms

Genetic algorithms are algorithms allowing for search area of alternative solutions to find the most optimal of them. The idea of genetic algorithms is derived from Darwin's theory of evolution, where the best individual replacing individual weaker adapted. Each individual has its chromosome, which is actually coded, in the form of characters (genes), point of the problem domain. Currently, the most commonly used is the binary notation. Group creates a population of individuals [Fig. 1.].

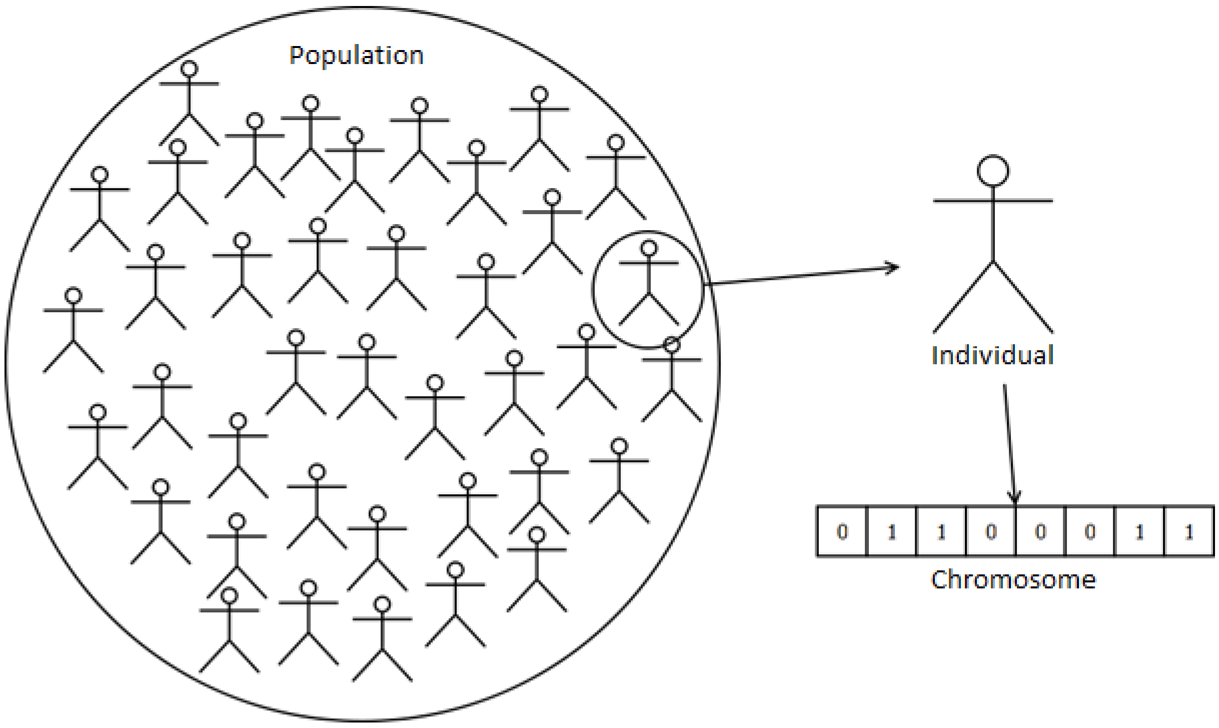

Fig. 1. *Population. Each individual has a different chromosome.*

Individuals, as well as in the environment, may crossover among themselves to form next generations. The chance to participate in reproduction, that is, creation of an individual in the new population is directly proportional to the adaptation (value of fitness function which determining how much the result satisfies our requirements). During the reproduction take place crossover (exchange fragments of genotypes) and sporadic mutations (random changes in the gene). Then are selected the best individuals from the two populations, or only individuals from the intermediate population. Diagram of a simple genetic algorithm is shown in [Fig. 2.].

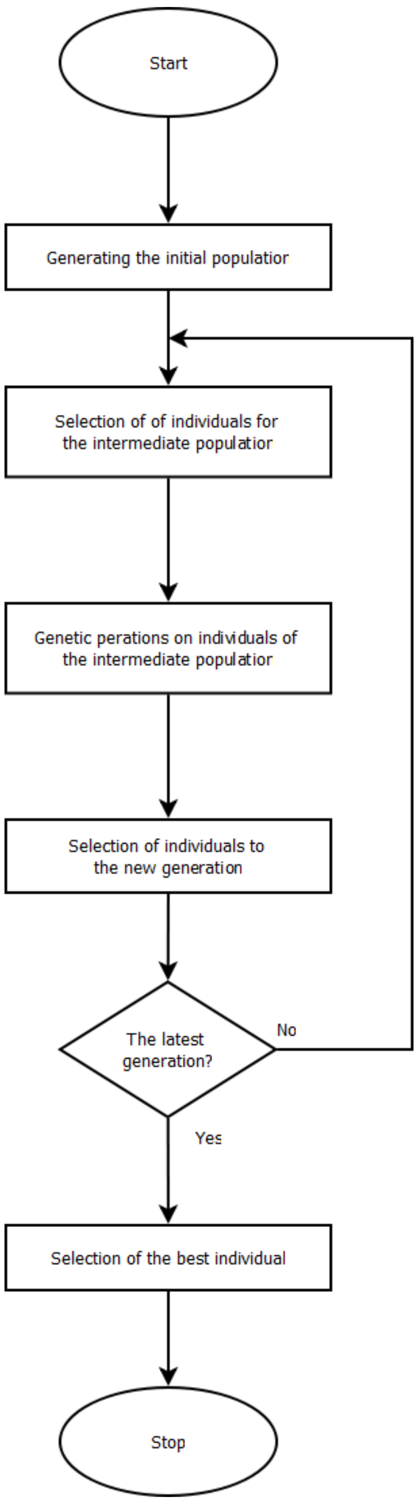
Start
Generating the initial populatior
Selection of of individuals for the intermediate populatior
Genetic perations on individuals of the intermediate populatior
Selection of individuals to the new generation
The latest generation?
No
Yes
Selection of the best individual
Stop

Fig. 2. *The working principle of a genetic algorithm*

3. CUDA technology and CUDA C language

In 2007 the NVIDIA company introduced the CUDA technology. It allowed developers direct access to dozens or even hundreds of cores placed on graphics cards and therefore easy to perform many advanced numerical computation on them [10]. Cores called threads are grouped into blocks that create grids. Both blocks and threads can be presented in one, two and three-dimensional area [Fig. 3.].Shared memory is a fast memory type of CASH. It is assigned to a specific block, and only the threads inside it have access to it. It is characterized by high speed, but small size compared to the global memory (as it was written in the introduction in GT200 architecture it is 16 KB).

CUDA C language brings to the classic C separation of functions into three types:

- host: functions called by the CPU and executed on the CPU,
- device: functions called by the GPU and implemented on the GPU,
- global: functions called by the CPU but the GPU performed, require the declaration of number of blocks and threads involved in the processing.

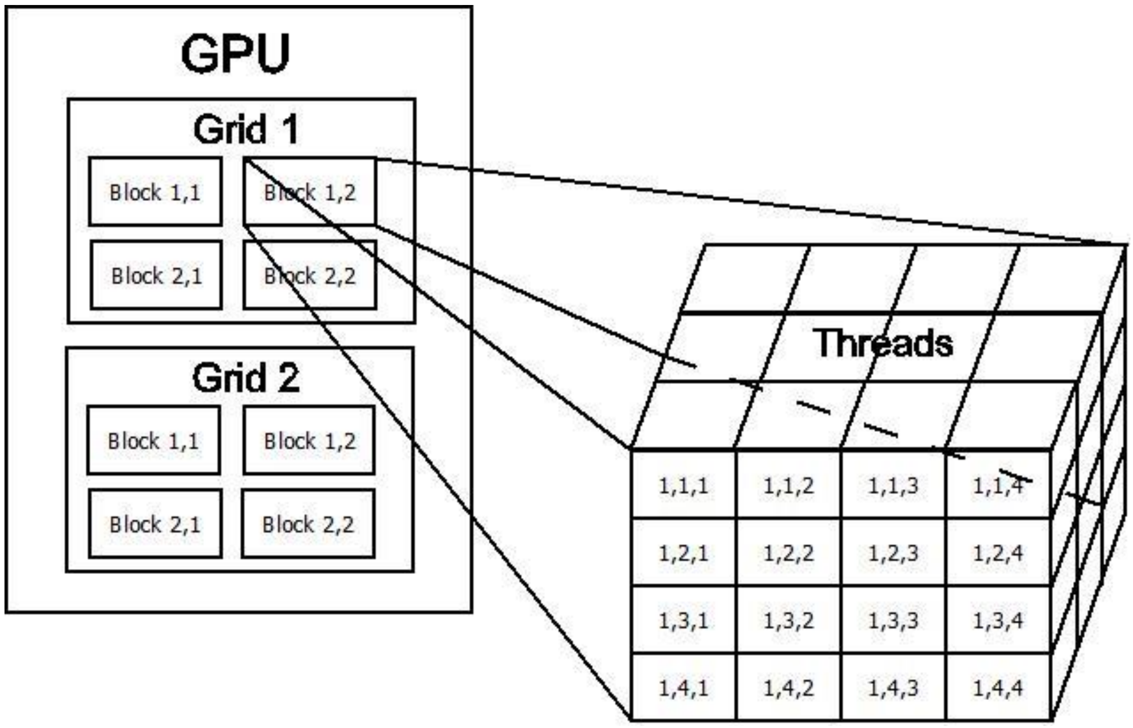

Fig. 3. *Division GPU on grids, blocks and threads.*

4. Projekt

In my work I have tried to concentrate on the possibilities of implementation a genetic algorithm with a large population and extensive genotype. The population can not be kept in the shared memory but only the portions, on which we operate at a time. The CUDA architecture does not have a built-in blocks synchronization, which brings with it synchronization problem after each step of the algorithm. There are three solutions:

- limit calculations to a single block,
- create a programming synchronization,
- synchronization management by the CPU.

In the first case there is a large reduction in the number of threads actively involved in the processing. This is due to the restrictions on the maximum number of cores per block in different architectures, for example, at GeForce 580 which is representative of Fermi architecture and have 512 cores involved in the performance of the algorithm could take only the 32 of them.

Software synchronization is relatively difficult to implement. It is based on observation and overwritten by each of the blocks the state array in the globa memory. In this way, they could check the status of the others and at the same time inform them of their own progress. This method, however, increases the load on the graphics card (especially the number of read operations performed on the global memory). Another problem is the inability to change the number of blocks and threads participating in the calculation of the kernel function (they can not be optimally chosen for each of the elements of the algorithm). Although this method will be verified in further work due to a suspicion of increase in productivity caused by resignation from synchronization by the CPU.

Management by CPU consists of division algorithm after which it is necessary to synchronize it. Each component is coded in the form of an independent kernel functions. After the kernel function is called cudaDeviceSynchronize function that causes suspend execution the main thread of the program until the end of the calculations on the graphics card. Then is called the next kernel function. Unfortunately, this method is largely impossible parallel execution of the application on the main thread, and at the same time on the video card, but it is not required in this type of applications.

In my work I used the test function of 30 variables, in which I was looking for a global maximum. I used division shown on [Fig. 4.].

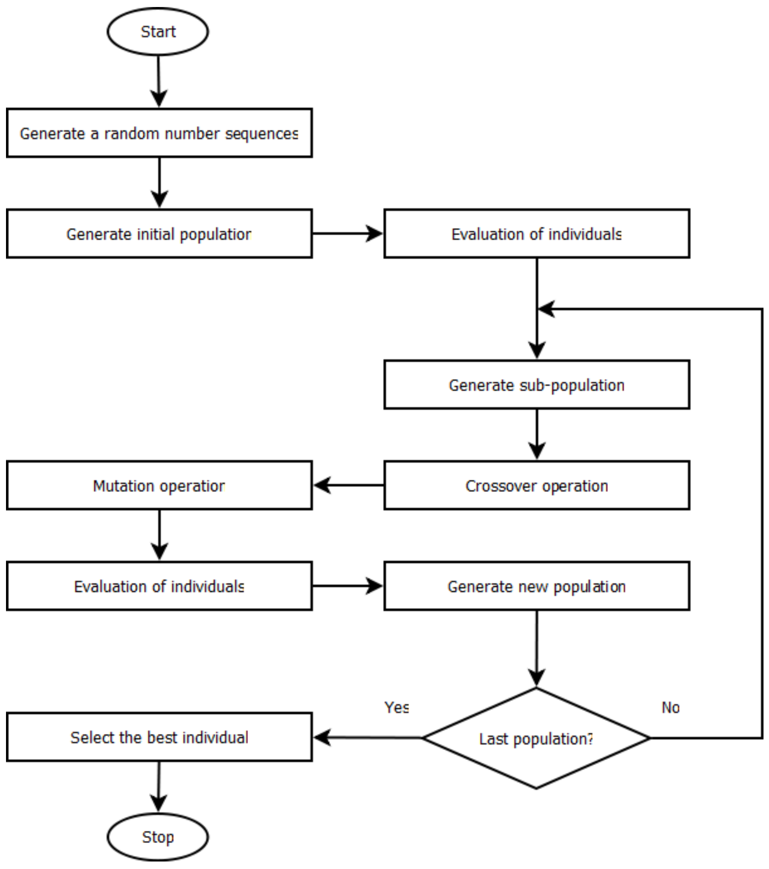
Start
Generate a random number sequences
Generate initial populatior
Evaluation of individuals
Generate sub-population
Crossover operation
Mutation operatior
Evaluation of individuals
Generate new population
Last population?
Yes
No
Select the best individual
Stop

Fig. 4. *A genetic algorithm used in the test*

This scheme includes only a part of the program execute on the graphics card. Blocks represent a single kernel functions, between which the CPU synchronizes and sets the appropriate number of blocks and threads. In the case of operations performed on genes of all individuals is set the number of threads equal to the length of the genotype and the number of blocks equal to the number of individuals in the population. This simplifies the construction of the code and allows to avoid, in some places, the need of a complicated calculation of area of population for which the thread is responsible. In other cases is set the maximum effective number of threads and blocks (64 threads, which is the optimal number of threads per block [9]). [Table 1.] shows this division.

Table 1. - *Number of allocated threads and blocks depending on the function of kernel*

Division of threads	Topic
Number of threads = length genotype Number of blocks = size of the population	Generate a random number sequences
	Generate initial population
	Crossover operation
	Mutation operation
Number of threads = 64 Number of blocks = (number of threads involved in the processing + 1) / 64	Evaluation of individuals
	Generate sub-population
	Generate new population

5. Results

The results are quite satisfactory. There was a significant increase of acceleration of a genetic algorithm calculation in comparison with the calculation performed on the processor [Tabela 2]. In the paper was taken into account the time needed for memory allocation and copying of data between the CPU and GPU because in many programs it is part greatly affects the speed of the algorithm.

Table 2. - *The results obtained on the CPU and selected GPUs*

Procesor (Intel Core i5-2430M)	GeForce GT 540M	GeForce 460 GTX
17560ms	5651ms	1926ms

References

[1] Cano Alberto, Zafra Amelia, Ventura Sebastian, Speeding up the evaluation phase of GP classification algorithms on GPU, SOFT COMPUTING 16, 2012, p. 187-202

[2] Munawar Asim, Wahib Mohamed, Munetomo Masaharu, Hybrid of genetic algorithm and local search to solve MAX-SAT problem using nVidia CUDA framework, GENETIC PROGRAMMING AND EVOLVABLE MACHINES 10, 2009, p.391-415

[3] Wong Man-Leung, Wong Tien-Tsin, Parallel Hybrid Genetic Algorithms on Consumer-Level Graphics Hardware, 2006 IEEE CONGRESS ON EVOLUTIONARY COMPUTATION, 2006, p.2958-2965

[4] Harding Simon, Banzhaf Wolfgang, Fast Genetic Programming on GPUs, LECTURE NOTES IN COMPUTER SCIENCE, 4445, p.90-101

[5] Man Leung Wong, Tien Tsin Wong, Implementation of Parallel Genetic Algorithms on Graphics Processing Units, Studies in Computational Intelligence 187, 2009, p.197-216

[6] Shen Zhen, Wang Kai, Zhu Fenghua, Agent-based Traffic Simulation and Traffic Signal Timing Optimization with GPU, IEEE International Conference on Intelligent Transportation Systems-ITSC, 2011, p.145-150

[7] Vidal Pablo, Alba EnriqueA, Multi-GPU Implementation of a Cellular Genetic Algorithm, IEEE Congress on Evolutionary Computation, 2010

[8] Maitre Ogier, Querry Stephane, Lachiche Nicolas, EASEA Parallelization of Tree-Based Genetic Programming, IEEE Congress on Evolutionary Computation, 2010

[9] Arora Ramnik, Tulshyan Rupesh, Deb Kalyanmoy, Parallelization of Binary and Real-Coded Genetic Algorithms on GPU using CUDA, IEEE Congress on Evolutionary Computation, 2010

[10] Harish Pawan; Narayanan P. J., Accelerating large graph algorithms on the GPU using CUDA, HIGH PERFORMANCE COMPUTING - HIPC 2007, 2007, p.18-21

[11] Gutierrez Eladio, Romero Sergio, Trenas Maria A, Simulation of quantum gates on a novel GPU architecture, PROCEEDINGS OF THE 7TH WSEAS INTERNATIONAL CONFERENCE ON SYSTEMS THEORY AND SCIENTIFIC COMPUTATION, 2007, p.121-126

[12] Manavski Svetlin A.CUDA COMPATIBLE GPU AS AN EFFICIENT HARDWARE ACCELERATOR FOR AES CRYPTOGRAPHY, 2007 IEEE INTERNATIONAL CONFERENCE ON SIGNAL PROCESSING AND COMMUNICATIONS, 2007, p.65-68

[13] Belleman Robert G.; Bedorf Jeroen; Zwart Simon F. Portegies, High performance direct gravitational N-body simulations on graphics processing units II: An implementation in CUDA, NEW ASTRONOMY, 13, 2008, p.103-112