Implementation of the genetic algorithm by means of CUDA technology involved in travelling salesman problem

Anna Plichta
Tomasz Gąciarz
Cracow University of Technology, Department of Computer Science
31-155 Cracow, ul.Warszawska 24
Email: aplichta@pk.edu.pl
Email: tga@pk.edu.pl

Bartosz Baranowski
Email: bbaranowski@onet.pl

Szymon Szomiński
AGH University of Science and Technology
30-059 Cracow, Al. A. Mickiewicza 30
Email: szsz@agh.edu.pl

KEYWORDS

genetic algorithm, CUDA technology, travelling salesman problem

ABSTRACT

The research was intended to solve the travelling salesman problem by means of genetic algorithms. The implementation of the algorithm was by virtue of CUDA technology. The research was focused on checking how much the system can improve if instead of classical CPU processors one uses GPU graphical processors enabled to perform the operations parallel. The algorithm was implemented in the high level CUDA C language. Thus, measuring the pure time of performance of the algorithm could be the single but reliable point of comparison between two above mentioned types of processors. Making some operations mutually independent and using CUDA technology makes the task much faster to execute. Due to it complex issues can be solved in a shorter time.

INTRODUCTION

Video cards were primarily used to display graphics, but it was quickly observed that GPU processors may be much faster than CPU on one condition: the problems to sort out should be parallel. Hence, more and more supercomputers of the TOP500 list are based on NVIDIA Tesla video cards and many so called GPGPU technologies were developed (General-Purpose Computing on Graphics Processing Units) which enable for making general computations by virtue of video card processor. The most popular of them are NVIDIA CUDA (Compute Unified Device Architecture), OpenCL (Open Computing Language) and Microsoft DirectCompute.

The aim of the project was to verify CUDA technology and GPU graphic processors in respect of paralell genetic algorithms. Therefore, the genetic algorithm was implemented into the traditional CPU processor-based technology and into GPU graphic processor-based technology in order to compare the efficiency of both implementations. The travelling salesman problem was chosen to check the genetic algorithm.

GENETIC ALGORITHM

Genetic algorithm (GA) by John Holland and David E. Goldberg [4], [2] is the adaptation searching procedure based on the mechanisms of natural selection and natural genetics. These paralell algorithms are simple, general and efficient. Genetic algorithms (GA) can be applied to designing electric machines and circuits, to optimizing routes, to lofting in telecommunication or in computer games. Genetic algorithms can be also used to solve NP problems such as travelling salesman problem [7], [8], [9]. Travelling salesman problem (TSP) pertains to the graph theory and it consists in finding minimal Hamilton cycle in the full weighted graph. Usually, the issue is represented from the point of view of the travelling salesman who has to visit N cities (each city once). The distances between all cities are known. The problem is NP-difficult [16].
As for GA, points belonging to the search space are represented as limited-length chains of limited-alphabet symbols. In order to solve the problem GA (in each generation) transforms the set of points of search space (called chromosomes). That particular collection is called population.

Chromosomes within the particular population compete one another for survival and possibility of reproduction. During each iterative step of the genetic algorithm called geneara-

Proceedings 28th European Conference on Modelling and Simulation ©ECMS Flaminio Squazzoni, Fabio Baronio, Claudia Archetti, Marco Castellani (Editors)
ISBN: 978-0-9564944-8-1 / ISBN: 978-0-9564944-9-8 (CD)

tion all individuals are evaluated and undergo recombination according to their level of usefulness and some ”genetic operators”. As a result, the following populations are growing better and better. That process is usually suppressed when the desired number of populations is achieved or when the population ceases to change [10].

The function evaluating each chromosome which is called by biologists the fitness function is a measure of benefit or profit we want to maximize. The first genetic operator called reproduction works according to the values of the above-mentioned function. During that process some chromosoms are selected to undergo the next operation (the more adapted the chromosome the higher its chance for selection). All selected chromosomes are coupled randomly. During the crossover process (the second genetic operator) they exchange some elements in order to create new offspring. The third genetic operator called mutation is used in order to avoid premature convergence to one of the local minima and increase the diversity of the population. It changes the values of some content of the chromosome at random consequently making up its almost identical copy [2].

In order to solve the problem with the genetic algorithm, the latter should be represented as chromosome. Moreover, the crossover and mutation should be prepared as well as the fitness function which measures the usefulness of the chromosome regarding the issue.

GA AND THE TRAVELLING SALESMAN PROBLEM

Encoding of the chromosome

For the purpose of travelling salesman problem where each individual represents one route the most common encoding is encoding as integers. Each city has its unique number and appears in the chromosome only once. In the Figure 1 the route starts in the point (city) 3 and leads through points 1,2,5 and 4. In many cases the route is additionally represented as a loop. The additional edge from the point 4 to 3 appears [5].

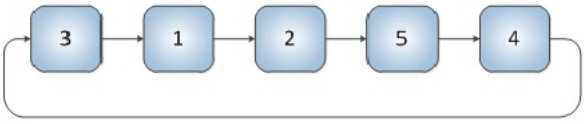

Fig. 1. The example of encoding the chromosome with integers

Genetic operators - selection, crossover, mutation

Two methods of selection were implemented and tested, namely, the method called roulette wheel and the tournament one. Practically, the latter proved to be more efficient.

One should modify the standard crossover procedure so that to take account of the structure of the chromosome adapted specifically to the travelling salesman problem. Three crossover methods were implemented and tested: one-point crossover, two-point crossover and greedy crossover.

However, because in the travelling salesman problem each gene is represented by the unique integer which cannot appear in the indivual more than once, one should add some correcting mechanisms. Just as in the binary representation the first step is to copy the left side of the first parent into the offspring. Then only these genes from the right side of the second parent are copied which the offspring lacks. Finally, one should find and copy the lacking genes into the offspring. For instance, the second parent can be searched from its left side until a lacking gene is found.
Two-point crossover is more advanced version of the one-point crossover. In two-point crossover both parents are cut in two random points in order to get left, right and the middle part. The offspring is joined together from the middle section of one parent and the extreme parts of the second parent. Just as in one-point crossover one can simultaneously create two descendants and the correcting mechanisms should be added to prevent the descendant from having two or more identical genes [19].
First of all, the middle part of the first parent is copied into the descendant. Then, the genes which the descendant lacks are copied from the extreme parts of the second parent. At last, the descendant is given the genes which it still lacks. For instance, the second parent can be searched from its left side until a lacking gene is found.
Greedy crossover is the method applied specifically to the travelling salesman problem (or similar ones) as it can be used only in case each gene in the chromosome in unique and appears only once within it [1].

For the purpose of the travelling salesman problem mutation is usually changing the places of two random gene of the descendant. The occurence of the mutation is infrequent (its probability is about 0,5%).

The aim function

In the project we applied simple adaptation function consisting in computing the length of the route of the individual. The better (shorter) the route the lesser the value of the adaptation function. Distances between the cities are stored in two-dimension table.

Aiding mechanisms

2opt Heuristics
In order to improve the results in many versions of genetic algorithms one uses also heuristic methods [21]. For instance, 2opt method is often used to solve the travelling salesman problem. It is a method of local optimization consisting in reconfiguring two edges so that to achieve locally shorter route.

IMPLEMENTATION INTO CPU PROCESSOR

To implement the algorithm into the CPU processor the Code::Blocks environment was used together with MinGW compiler (v.4.6.1) based on GCC. In order to provide multithreading the OpenMP library was used.
The C language was chosen mainly because such choice facilitates the implementation of the algorithm into graphic card in CUDA C language in the second stage of the research. The implementation of the algorithm underwent many changes. Initially, it was a simple genetic algorithm provided with one-point crossover and mutation. The selection method was roulette wheel.

In the next version the selection method was replaced with the tournament one and one-point crossover was replaced with two-point crossover. In the next version greedy crossover appeared and additional heuristic methods which improved achieving good results faster. At last, the island model was provided.
The final version of the algorithm was comprised of tournament selection, greedy crossover and standard mutation. The results were also improved by means of the 2opt method. Multithreading was achieved thanks to adding islands served by separate threads. In each island there is a separate instance of the genetic algorithm, served by the separate plot. Instances communicates one another by means of fixed-frequency migration. Each island sends and receives some of its best individuals. The number of migrating individuals depends on the size of the population.

There are two conditions of ceasing the algorithm in the implementation. The program ceases to work if the fixed number of iterations of the main algorithm loop is exceeded. The second condition checks if in a few hundred generations the improvement of the results is relevant enough. If not, the algorithm states that the optimal individual was created and ceases to work.

IMPLEMENTATION INTO GPU PROCESSOR

In order to implement the algorithm into the GPU processor the Microsoft Visual Studio 2008 was used together with NVIDIA CUDA Toolkit 4.0. During the implementation of the algorithm in the CUDA C language (consequently, into the graphic card with its architecture) the algorithm had to undergo some changes. To work efficiently on the GPU processor the program must have high level of parallelity. According to the golden rule by CUDA programmists, there should be 24 times more plots than cores in the graphic card to use the graphic card efficiently. The algorithm was basically written for the NVIDIA GeForce GTX 285. graphic card (it has 240 cores). Hence, the algorithm was transformed to use min. 5760 plots [13], [19].
In comparison to the version for the CPU processor there was one significant change: each individual is served by another thread. The population of 100 individuals results in just 100 threads, which is far too small for CUDA technology. Hence, to increase the number of threads the additional changes were introduced. First of all, the number of indivuals reached 5760. However, the tests the CPU version of the algorithm had undergone proved that increasing the number of islands improves the results more than increasing the number of individals. In order to divide the 'genetic material' between the islands we used the standard CUDA technology mechanism splitting threads into blocks. 60 threads (100 individuals in each) gives 6000 threads which is suaitable for CUDA technology for GTX 285 graphic card [6], [18]. To sum up, the entire population was divided into M islands each having N individuals. Each individual is served by the separate thread. Blocks communicate one another by virtue of global memory of the graphic card. Inside the blocks the population is stored in shared memory which is a little bit faster.
The code dedicated to the graphic card is encrypted in kernels which are separate functions. They are executed fully in the GPU processor. Kernels can use only the data kept in the graphic card memory. That memory is allocated by means of cudaMalloc function. cudaMemcpy() function is responsible for transferring data between RAM and graphic card memory (and vice versa).

TESTS

Test implemantation dedicated to the CPU processor was carried out on the processor Intel Core 2 Duo E8400 (3600 Mhz) provided with double physical cores and 6MB second level cache.
The implementation dedicated to the GPU processor was carried out on NVIDIA GeForce GTX 285 graphic card provided with 240 cores and 1024 MB of GDRR3 memory (702 Mhz). To ensure that the measurements are credible in both implementations time was measured in milliseconds.

The implementation dedicated to the CPU processor was run on one thread working on one physical core and on two threads working on two physical cores. In the latter case (2 cores) the population was divided into two islands. Measurement results for the 10,100, 1000 and 10000 individuals are presented below in the Figures: 2,3,4,5.
Each measurement was repeated ten times. The mean of these ten measurements was taken into account (fig. 6). Regarding the implementation dedicated to the CPU processor, to measure the time we use the QueryPerformanceCounter() function. In the version dedictated to the GPU processor the cudaEventRecord() function was used. The length of chromosomes (number of the cities) was the same for both CPU and GPU experiments (excluding the minimal case of 10 cites) and was equal to 100.

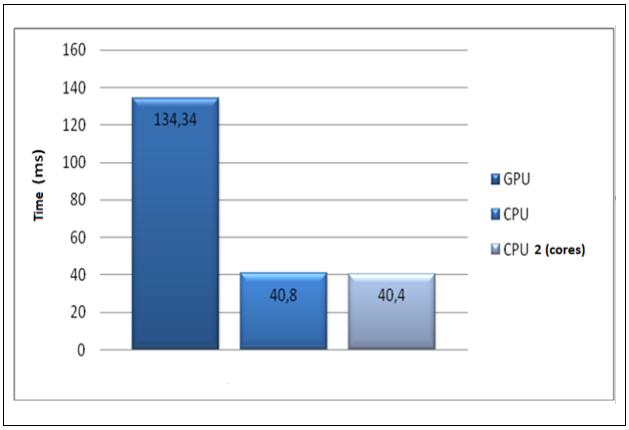

Fig. 2. Measurement results for: the length of the chromosome 10; size of the population 10; number of islands 1; number of generations 100

The time the CPU processor needs for computations raise almost linearly when the amount of data increases but when the graphic card is used it hardly changes. It proves that the graphic card needs a vast number of threads to reach its full efficiency. The GPU processor is not efficient when the number of threads is small because the majority of the cores is unused.

For the sake of the tests the set of points was created consisting of 1024 cities. Each city (point) is given the number from 0 to 1023 as well as carthesian system of

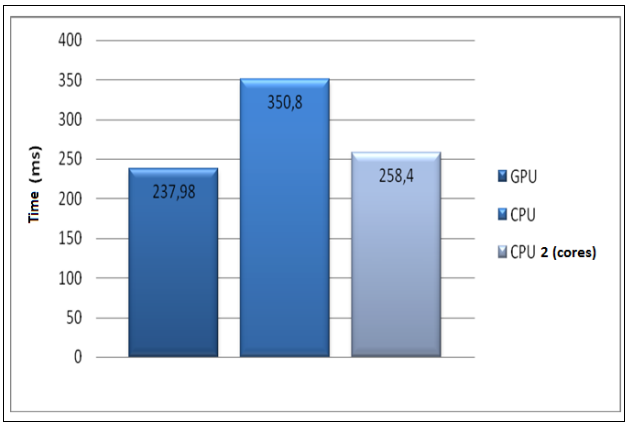

Fig. 3. Measurement results for: the length of the chromosome 100; size of the population 100; number of islands 1; number of generations 100

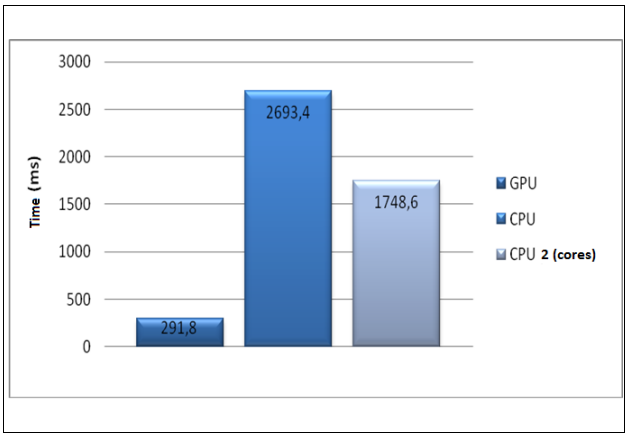

Fig. 4. Measurement results for: the length of the chromosome 100; size of the population 1000; number of islands 10; number of generations 100

coordinates x and y. Depending on the chromosome length, always the first N points is used where N stands for the chromosome length. Additionally, in order to enable for the comparison of the effects of the working algorithm, three sets of points taken from TSPLIB library were added. The following sets were chosen berlin52 (52 points), kroA100 (100 points) and kroA150 (150 points)[17].

According to the tests we carried out, graphic cards can be a very good alternative to the standard CPU processors but the problems to solve should be possible to paralell. Moreover, at the small number of threads the efficiency of the both above-mentioned methods are barely distinguishable. Graphic cards require a few dozen of threads to be fully efficient. Using ten thousend of threads enabled us to achieve the acceleration 23-times improved in relation to one CPU core or 13-times improved in relation to two CPU cores.

SUMMARY

The principles of the project were realized. Two versions of the genetic algorithm were created. The first was dedicated to the CPU processor and the second to the graphical processor

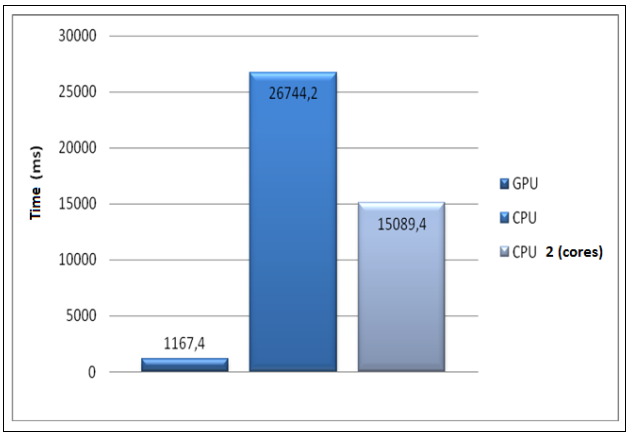

Fig. 5. Measurement results for: the length of the chromosome 100; size of the population 1000; number of islands 100; number of generations 100

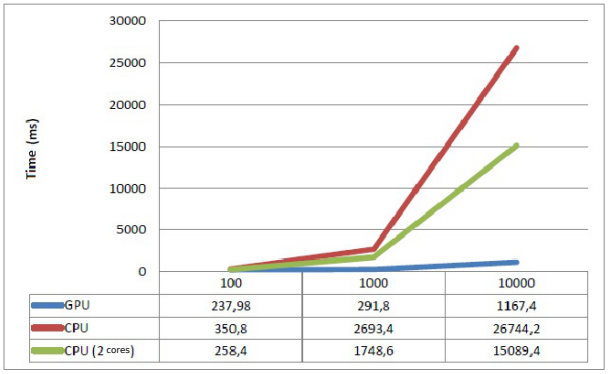

Fig. 6. Time algorithms depending on the size of the population

GPU. The algorithm is able to solve the travelling salesman problem even for 200 cities which is equal to about 7,89 x 10^{374} possible combinations of the values of genes. The version dedicated to the CPU processor was made parallel by virtue of the OpenMP library. The implementation dedicated to the graphic card was made by means of the CUDA technology. We succeeded in achieving the acceleration 23-times improved in relation to one CPU core and the acceleration 13-times improved in relation to two CPU physical cores. It proves the dormant potential of the contemporary graphic cards. For the parallel computations they became the significant alternative for the traditional CPU processors. Hence, the CUDA technology is more and more often used in supercomputers.

REFERENCES

[1] Soushil L. J. and Gong L., „Augmenting Genetic Algorithms with Memory to Solve Travelling Salesman Problems", *University of Nevada, Reno*,2007.

[2] Holland J. H., „Adaptation in Natural and Artificial Systems", *MA:MIT Press, Cambridge*,1992.

[3] Mitchell M., „An introduction to Genetic Algorithms", *A Bradford Book, Londyn*,1999.

[4] Goldberg D. E., „Genetic algorithms and their applications", *WNT*,2003.

[5] Razali M. N. and Haraghty J., „Genetic Algorithm Performance with Different Selection Strategies in Solving TSP", *WCE*,2011.

[6] Sanders J. and Kandrot E., „CUDA example", *Helion*,2012.

[7] Goldberg D. E., „Genetic Algorithms in Search, Optimization and Machine Learning", *Addison-Wesley Longman Publishing Co.*,1989.

[8] Lawrence D., „Handbook of genetic algorithm", *Van Nostrand Reinhold*,1991.

[9] Kim K. and Man F. and Tang and K.S. Kwong and S., „GENETIC ALGORITHMS.: Concepts and Designs Avec disquette Advanced Textbooks in Control and Signal Processing Series", *Springer*,1999.

[10] Sivanandam S.N. and Deepa S.N., „Introduction to Genetic Algorithms", *Springer-Verlag Berlin Heidelberg*, 2007.

[11] Coley David A., „An introduction to genetic algorithms for scientists and engineers", *World Scientific*,1999.

[12] Michalewicz Z., „Genetic Algorithms + Data Structures = Evolution Programs. Artificial Intelligence Series", *Springer-Verlag*,1992.

[13] NVIDIA Corporation, „cuda toolkit documentation", http://developer.download.nvidia.com,2011.

[14] Camilo Rostoke, „Travelling Salesman Problems",http://top500.org/list/2011/11/100,2012.

[15] Camilo Rostoke, „Travelling Salesman Problems",http://www.cs.ubc.ca/labs/beta/Courses/CPSC532D-05/Slides/tsp-camilo.pdf,2012.

[16] GPGPU, „General-Purpose Computation on Graphics Hardware",http://gpgpu.org/developer,2012.

[17] Open MP.org, „The OpenMP API specification for parallel programming",http://openmp.org/wp/openmp-specifications/,2012.

[18] Marek Obitko, „Introduction to Genetic Algorithms",http://www.obitko.com/tutorials/genetic-algorithms/crossover-mutation.php,2012.

[19] Konstantin Boukreev, „Genetic Algorithms and the Travelling Salesman Problem",http://www.codeproject.com/Articles/1403/Genetic-Algorithms-and-the-Traveling-Salesman-Prob,2012.

[20] SENGOKU, H. YOSHIHARA, Ikuo, „A Fast TSP Solver Using GA on JAVA",http://www.cs.us.es/cursos/ia1-2006/trabajos/arob98.pdf,2012.

[21] Brainz.org, „15 Real-Word Uses of Genetic Algorithm",http://brainz.org/15-real-world-applications-genetic-algorithms,2012.

[22] forums nvidia com, „forums nvidia com", http://forums.nvidia.com/index.php?showtopic=195749,2012.

[23] cs helsinki fi, „cs helsinki fi bresenh", http://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html,2012.

[24] econ iastate edu, „econ iastate edu ",http://www2.econ.iastate.edu/tesfatsi.html,2012.

AUTHOR BIOGRAPHIES

ANNA PLICHTA She studied comparative literature at the Jagiellonian University and obtained her degree in 2007. She also studied computer science at Cracow University of Technology and obtained her degree in 2010. Currently, she works as a teaching fellow at Cracow University of Technology. Her e-mail address is: aplichta@pk.edu.pl

TOMASZ GĄCIARZ was born in Olkusz. He studied computer science at the AGH University of Science and Technology. and obtained her degree in 1994. Currently, he works as a teaching fellow at the Cracow University of Technology. His e-mail address is: tga@pk.edu.pl

BARTOSZ BARANOWSKI I was born in Cracow. He studied computer science at the Cracow University of Technology and obtained his degree in 2010. Currently, he works at the IT company. His e-mail address is: bbaranowski@onet.pl

SZYMON SZOMIŃSKI He studied computer science at the Cracow University of Technology and obtained his degree in 2010. Currently, he study computer science at the AGH University of Science and Technology. His e-mail address is: szsz@agh.edu.pl