GENETIC ALGORITHM ON GENERAL PURPOSE GRAPHICS PROCESSING UNIT: PARALLELISM REVIEW

A.J. Umbarkar[1], M.S. Joshi[2] and N.M. Rothe[3]
[1]*Department of Information Technology, Walchand College of Engineering, India*
E-mail: anantumbarkar@rediffmail.com
[2]*Department of Computer Engineering, Pune University, India*
E-mail: madhuris.joshi@gmail.com
[3]*Department of Computer Science and Engineering, Walchand College of Engineering, India*
E-mail: nmrothe.11189@gmail.com

Abstract
Genetic Algorithm (GA) is effective and robust method for solving many optimization problems. However, it may take more runs (iterations) and time to get optimal solution. The execution time to find the optimal solution also depends upon the niching-technique applied to evolving population. This paper provides the information about how various authors, researchers, scientists have implemented GA on GPGPU (General purpose Graphics Processing Units) with and without parallelism. Many problems have been solved on GPGPU using GA. GA is easy to parallelize because of its SIMD nature and therefore can be implemented well on GPGPU. Thus, speedup can definitely be achieved if bottleneck in GAs are identified and implemented effectively on GPGPU. Paper gives review of various applications solved using GAs on GPGPU with the future scope in the area of optimization.

Keywords:
Genetic Algorithm (GA), Parallel Genetic Algorithm (PGA), General Purpose Graphics Processing Unit (GPGPU), Compute Unified Device Architecture (CUDA), Open Computing Language (OpenCL)

1. INTRODUCTION

The GA is one of the most important soft computing tools used to solve many engineering optimization problems. Being a soft computing tool, the evolution of a solution in GA is a non-deterministic process [29]. Hence, many times, while working on GA, one needs to deal with problem of convergence or premature convergence. Though the single population GA has shown excellent search performance but performance can be improved by increasing the population size or having more than one population. The latter one provides more diversified search to find global optimum solution. This is called as diversity problem of GA. Another way to explore search space is to increase the population size. But when we increase the population size the search space is explored to greater extent and this very large size of population deteriorates the performance of evolutionary process of GA. This is called as population size problem. Also, the GA has shown excellent search performance when applied to small (5-10) dimension problems, but many GAs suffer from the curse of dimensionality problem when applied to large dimensionality problems. The curse of dimensionality problem means it deteriorates performance quickly as the dimension of search space increases.

To tackle all these problems, mentioned above, one can use the GPGPU and can exploit its functionality to solve GA effectively with more speed up.

Nowadays GPGPU technology is getting cheaper and common; one cannot ignore its importance anymore. To enhance GA's performance, GA can take advantage of parallel computing environment based on its well paralleled nature [34]. The GPGPU based GAs try to resolve the convergence problem or premature convergence, diversity problem, population size and curse of dimensionality problem.

Also many real-life problems may need days or weeks of computing time to solve on serial machines. Although the intrinsic time complexity of a problem cannot be lowered by using a finite amount of computing resources in parallel, parallelism often allows reducing time to reasonable levels. This is very important in an industrial or commercial setting where the time to find the solution is critical for decision making.

Nowadays there is a need of performance improvement in each application so as to minimize the time required for execution. The graphics processing unit (GPU) based computing is a broad area under which highly computational problems are solved. The GPU attracted many researchers because of its low-cost, high-performance computing and high availability. Though the GA's algorithmic development is at extreme level, but there is full scope for making it parallel on GPUs and its performance can be improved by minimizing data transfer between a CPU and a GPU by executing its operations of evaluation, selection, and reproduction through GPU.

This paper provides brief literature review of implementation of GAs on GPGPU. In the remnant of this paper we briefly describe CUDA language and GPGPU architecture. Then we explain various optimization strategies for implementation of GAs on GPGPU. Thereafter we discuss recent work in the field of GA carried out by various researchers on GPGPU. Finally, we discuss the finding of this review work and future scope for further improvement and also give few guidelines for implementing GA on GPU.

2. GPGPU COMPUTING AND CUDA

In parallel computing the usage of GPU for general purpose computing is called as General Purpose GPU (GPGPU). Each GPU (also called as 'device') has multiple streaming multiprocessors (SM) inside it. Each SM has separate local memory shared by multiple streaming processors (SP), which also has separate register associated with it, shown in Fig.1.

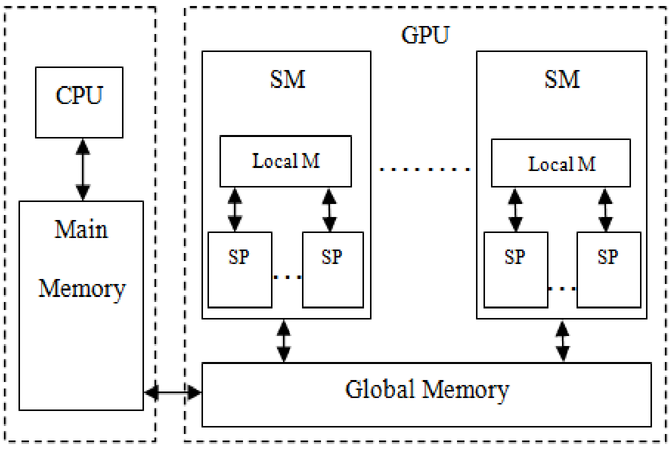

Fig.1. GPGPU Architecture

Thus, GPU has multiple processors which can execute simultaneously. Thus, GPU exhibits SIMD Flynn's taxonomy which describes computers having multiple processing elements which perform the same operation on multiple data points simultaneously. In this way, the GPU's multi-core architecture inherently supports data parallelization in SIMD form [31]. So there is huge scope for parallelization and thus, GA can be effectively parallelized on GPGPU to achieve better speedup. Thus, better optimal solution can be obtained in less amount of time. The trend of general-purpose computing on GPUs (GPGPU) may lead to wider use of SIMD in the future.

Today the GPUs of NVIDIA / Intel / AMD vendors are advanced and far ahead of the CPUs in terms of number of cores and their computational power.

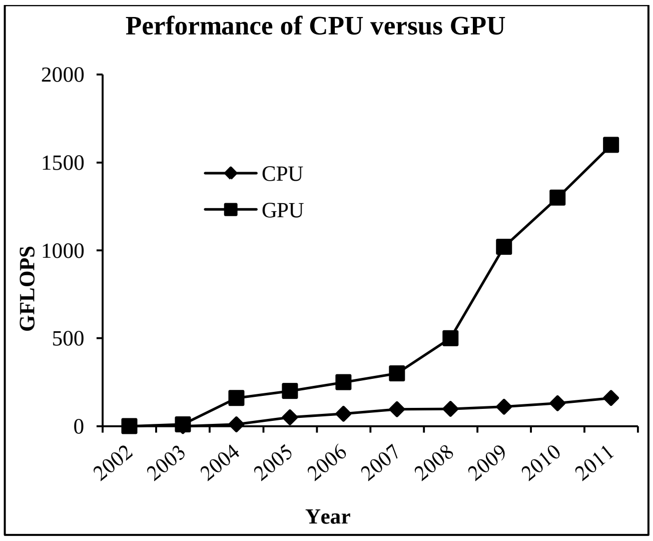

Fig.2. Performance of CPU versus GPU

The Fig.2 shows performance of CPU versus GPU. Though the GPU was available for general purpose applications but there was no high level language available till 2007 to write application for GPU. Therefore GPU was still popular among the graphics language users only.

In 2007, NVIDIA, the inventor of GPU took initiative and invented "Compute Unified Device Architecture" (CUDA) which is an API that made possible to use high level languages known, such as C, C++, DirectX, FORTRAN, Java and python, to write an application for GPU. With the launched of CUDA, GPU became available for every programmer. Add-ons for CUDA (such as Eclipse IDE) are also available nowadays. Apart from CUDA there are various APIs such as ATiCAL (Calculation Abstraction Layer), OpenCL (Open Compute Language) available which can be used to write general applications over GPU. There are more frameworks available nowadays but CUDA and OpenCL are mostly used.

The GPU operates as a coprocessor to the CPU (also called as 'host') which executes the program. The basic unit of a parallel code on CUDA is a thread, thousands of which can be deployed simultaneously on GPU card. Each thread has its own set of memory registers. With dynamic scheduling and fast creation and destruction of light-weight threads, CUDA is best suited for problems whereby same instruction set is to be executed on different data via multiple threads (SIMD execution) [30].

3. GA OPTIMIZATION ON GPGPU

The possible strategies for performance optimization of GAs/PGAs on GPU are:

Structure optimization:

- Structuring the GAs/PGAs to exploit maximum parallel execution and high arithmetic intensity of GPUs.
- Minimizing data transfers between the CPU and the GPU.

Programming optimization:

- Achieving global memory coalescing – Simultaneous memory accesses by a group of threads can be performed as a single memory transaction.
- Maximizing use of shared memory conflicts.
- Minimizing the number of local variables per thread.
- Minimizing branching of threads within a warp.
- Minimizing the need of explicit synchronization among threads.
- Another strategy is to have threads arranged into blocks. Where each block runs on one multiprocessor. It is also possible to have more blocks than multiprocessors and more threads per block than cores, to get optimal use of GPU.
- Shared memory may be accessible only within the block and thread synchronization is possible also only within the block.

Performance can also be optimized if parallelization of GA is done. The Way in which GAs/PGAs are parallelized depends upon following parameters [32]:

- How 'fitness' is evaluated.
- How Crossover / mutation is applied.
- Single or multiple subpopulation(s) are used.
- If multiple population are used, how individuals are exchanged.
- How selection is applied.

Depending on how each of these elements is implemented, there are more than ten different methods of PGAs. These can be classified into eight classes shown in Fig.3. Apart from first, all the remaining methods can be implemented on GPU. There are various GAs like SGA, steady state GA, binary and real-coded GA, cellular GA, HGA, adaptive resolution GA, also some PGAs such as massively parallel GA, hybrid master-slave GA, fine-grained parallel genetic algorithm (FGPGA), PMOEA / PMOGA implemented on GPU for few applications and it is found that remarkable speed up can be achieved. These GAs are implemented using various advanced programming languages such as MPI [33], OpenCL, DirectX and CUDA.

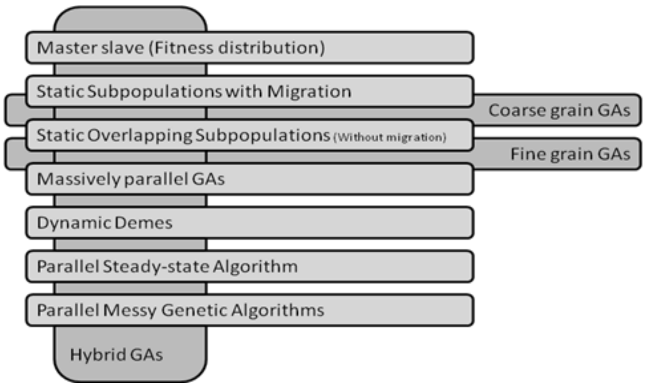
Master slave (Fitness distribution)
Static Subpopulations with Migration
Coarse grain GAs
Static Overlapping Subpopulations (Without migration)
Fine grain GAs
Massively parallel GAs
Dynamic Demes
Parallel Steady-state Algorithm
Parallel Messy Genetic Algorithms
Hybrid GAs

Fig.3. Eight classes of PGAs

4. LITERATURE REVIEW

Withayachumnankul, Laksanapanai and Pintavirooj (2004) [1] proposed hardware-accelerated objective function for medical image registration. The work was implemented using OpenGL AP on NVIDIA GeForce2 MX 200, RAM 32 Mbytes, AGP 2× card on CPU AMD 1.33GHz, SDRAM 512 Mbytes. The results for image interpolation requires half time less than it ever does without losing of accuracy.

Wong, Fok and Wong (2005) [2] proposed the implementation of PGA on GPU, here the hybrid of master-slave and fine-grained model is used which lead to the speedup factor of 1.25 to 5.02 for problems like feature selection, electrical circuit synthesis, and data mining.

Luo and Liu (2006) [3] implemented two common algorithms for SAT problems, first is greedy local search and second is GA, with some amendments for graphic hardware. The results obtained show a performance improvement on graphic hardware over ordinary CPU.

Wong and Wong (2006) [4] proposed parallel HGA on GPU with the Cauchy mutation operator and pseudo deterministic selection method. The proposed mutation and selection methods for function optimization gives excellent speed up factor ranging from 1.13 to 4.24 on GPU as compare to CPU.

Li, Wang, He and Chi (2007) [5] proposed fine-grained parallel genetic algorithm (FGPGA) on GPU. FGPGA based on GPU maps parallel GA algorithm to texture-rendering on consumer-level graphics cards. The proposed algorithm gives speedup and better results with increased population size as compare to FPGA.

Debattisti, Marlat, Mussi and Cagnoni (2009) [6] proposed simple GA on GPU (NVIDIA GeForce 8800 GT video card) using CUDA architecture for One-Max problem. Work emphasized on reducing the exchange of data with the CPU (Core2Duo processor at 1.86GHz).

Tsutsui and Fujimoto (2009) [7] implemented multiple population coarse grain GA on GPU (NVIDIA GeForce GTX285) for solving quadratic assignment problem. The speedup achieved is 3 to 12 times better than CPU (Intel i7 965 processor).

Zaloudek, Sekanina and Simek (2009) [8] accelerated significantly the evolution of cellular automata (CA) rules on GPU using GA. It is observed that, solving this problem on GPU using GA is speedy.

Nie, Lee, Han and Nie (2009) [9] proposed two measures to solve the general problems of GAs applied in stereo matching. First, the strategy of the simplified population-based incremental learning (PBIL) is adapted to decrease the problems in memory consumption and searching inefficiency. An alternative version of the proposed algorithm without using a probability vector is also presented for simpler set-ups. Second, to decrease the running time further, a model of the proposed algorithm can be run on programmable GPU is newly given. The result shows excellent speedup enhancement as compare to CPU implementation, in both, running speed and stability.

Robillard, Marion and Fonlupt (2009) [10] compared two schemes of parallelization useful for evaluation of several genetic programs. They proposed memory and representation optimization techniques that enhance computation speed up to 2.8 billion GP operations per second.

M. Wong and T. Wong (2009) [11] proposed to parallelize HGA on GPUs. They carried out experiments to compare parallel HGA with parallel fast evolutionary programming (FEP) and it is observed that former is more efficient than latter.

Arora, Tulshyan and Deb (2010) [12] proposed the parallelization of binary & real coded GA on GPU using CUDA. They parallelized the GA operators which gives a speed-up of 40 - 400 on the standard test problems.

Wong and Cui (2010) [13] proposed to solve data mining problems using parallel multi-objective evolutionary algorithm on GPU. The overall speedup is about 5.1 for data mining problems when compared with CPU based method.

Vidal and Alba (2010) [14] presented a multi-GPU implementation of a CGA using NVIDIA's CUDA technology. The speedup obtained on GPU (NVIDIA GTX-285 card) in comparison with CPU (Intel Quad processor at 2.67GHz) ranges from 8 to 771. The work demonstrates that multi-GPU desktops can serve a cost-effective parallel computing platform to obtain accurate results.

Kannan and Ganji (2010) [15] proposed GPU based GA to find the optimal docking position of a ligand to a protein. Proposed GPU based method gives 50x speedup on the fitness function evaluation and 10x to 47x speedup on the core GA.

Salwala, Kotrajaras and Horkaew (2010) [16] proposed the application of GPGPU to perform the simulations in emergent environment maps in gaming application. When serial version is compared with proposed method, the speed up of 11.79x is observed.

Yoshimi, Kurano, Miki and Hiroyasu (2010) [17] proposed a framework for implementation of parallel computing on GPU by evaluating simple genetic algorithm (SGA). Here they show the relationship between computational speed and execution condition.

Oiso, Yasuda, Ohkura, and Matumura (2011) [18] implemented Steady state GA on GPU using CUDA for function optimization. The results of steady state GA on GPU are 3 to 6 times better than CPU Intel corei7 (2.8 GHz).

Yongzhen, Yuhao and Dandan (2011) [19] proposed to solve image matching problem using GA on GPU. The results show the excellent performance improvement on GPU for image matching problem.

Sato, Hasegawa and Sato (2011) [20] observed that a practical solution can be obtained for Sudoku using GA on GPU within few seconds of processing time with a correct solution rate of 100%, even for extremely difficult problems by parallel processing of genetic computation on a GeForce GTX 460.

Munawar, Wahib, Munetomo and Akama (2011) [21] proposed adaptive resolution GA to solve non-convex mixed integer non-linear programming (MINLP) and non-convex non linear programming (NLP) problems over GPU. Paper enlists the challenges and design choices involved in parallelization of this algorithm to solve complex MINLPs over a commodity GPU using CUDA programming language. The result obtained over NVIDIA Fermi GPU shows speedup of up to 20x with double precision and up to 42x with single precision.

Bogdanski, Peter and Becker (2011) [22] proposed machine learning algorithm such as GA for parallelism on GPGPU. This algorithm dynamically chooses parameters for task scheduling and load balancing based on changing characteristics of the incoming workload for case study application as financial option pricing.

Kromer, Platos, Snasel, and Abraham (2011) [23] compared differential evolution (DE) and GA implemented on CUDA for solving the independent tasks scheduling problem. Their proposed approach seeks to utilize the resources of the GPGPU as much as possible and they found that DE is easy to implement using CUDA than GA.

Yoo, Harman and Ur (2011) [24] observed that it is relatively inexpensive to use GPGPU to run suitably adapted optimization algorithms, opening up the possibility of cheap scalability. They have developed a search based optimization approach for multiple objective regression test optimization, evaluated it on benchmark regression testing problems as well as larger real world problems and observed a speed–up of over 20x, using widely available standard GPUs.

Maitre, Kruger, Querry, Lachiche and Collet (2011) [25] presented EASEA by its algorithms and example problems. Speedup is observed on different NVIDIA GPGPU cards for various optimization algorithm families.

Lin, Igarashi, Mitani, Liao and He (2012) [26] presented a sensitive sketching interface that allows the user to interactively place a 3D human character in a sitting position on a chair. As reconstructing the 3D pose from a 2D stick figure is an ill-posed problem. They formulated this reconstruction into a nonlinear optimization problem and solved it using GA.

Wang and Shen (2012) [27] proposed artificial societies-computational experiments-parallel execution (ACP) approach in intelligent transportation systems (ITS) using parallel genetic algorithm. With GPU they generated activity plan for 1000 individual & household agent, checked its performance and observed speedup factor 23 for individual agent & speedup factor 32 for household agents.

Most of the implementation methods have achieved high performance, generally because they employed a population size larger than that generally employed in GAs in order to achieve a sufficient parallelization effect. This area is new and rapidly growing and there are various problems solved using GA on GPU, a bibliography of which is prepared by S. Harding [28] which is a list of 70 papers. In this paper, we summarize some important work. Table.1 gives the list of problems solved on GPGPU using GAs/ PGAs from year 2004 to 2011.

Table.1. Problems solved on GPGPU using GAs/ PGAs

SI. No.	**GA Type**	**Problem Solved**	**Year**
1	GA	Medical Image Registration[1]	2004
2	PGA- Hybrid Master-Slave and Fine-Grained Model	Feature Selection, Electrical Circuit Synthesis and Data Mining [2]	2005
3	CGA	SAT Problems[3]	2006
4	HGA	Function Optimization[4]	2006
5	FGPGA	Texture-Rendering[5]	2007
6	SGA	One-MAX Problem[6]	2009
7	GA	Quadratic Assignment Problems[7]	2009
8	GA	Cellular Automata Rules Acceleration[8]	2009
9	Population-Based Incremental Learning Based GA	Stereo Matching[9]	2009
10	GP	Benchmark Problems[10]	2009
11	PHGA	Benchmark Problems[11]	2009
12	Binary and Real-Coded GA	Function Optimization[12]	2010
13	PMOEA/ PMOGA	Data Mining[13]	2010
14	CGA	Function Optimization[14]	2010
15	GA	Drug discovery[15]	2010
16	GA	Gaming Application[16]	2010
17	GA	Benchmark problems[17]	2010
18	Steady-State GA	Function Optimization[18]	2011
19	GA	Image Matching[19]	2011
20	GA	Gaming-Sudoku Solution[20]	2011
21	Adaptive Resolution GA	Non-convex Mixed Integer Non-Linear Programming (MINLP) and Non-convex Non Linear Programming (NLP) Problems[21]	2011
22	GA	Task Scheduling and Load	

		Balancing Based on Workload and Case Study of Financial Option Pricing Problem[22]	2011
23	GA	Comparison of GA and DE[23]	2011
24	GA	MOEA Test Suite Minimization[24]	2011
25	GA	EASEA framework[25]	2011
26	GA	Sketching Interface[26]	2012
27	PGA	Daily Activity Plans[27]	2012

5. DISCUSSION

After reviewing the various papers of GAs / PGAs related to implementation on GPUs, we can say that:

- For solving various optimization problems using GAs with effective speedup, GPGPU is a good option.
- Along with parallelizing the task for solution, parallelization of genetic operators such as random number generation (RNG), selection, crossover mutation may also increase the effective speed up.
- RNG: CUDA libraries do not include any function of RNG at present, however there are some simple and fast implementation of RNG such as Xorshift [35], Park-Miller RNG [36] etc. available which make the process more effective.
- Selection: Being simple and easy to parallelize, tournament selection method is mostly preferred for parallel implementation of GA.
- Crossover and Mutation: Generally, single point crossover and mutation is preferred. For making these operators effective block size, warp size etc. need to be designed properly. Some implementation guidelines are given in [12, 37].
- Most of the GAs / PGAs are yet to be implemented on GPGPU. So researchers have huge scope in this area and can further exploit the functionality of GAs using GPU.
- Based on search space complexity of problems it is possible to provide diversity in search space using GAs / PGAs on GPGPU.
- Many real life problems, those support the SIMD (Flynn's taxonomy) architecture, can be effectively solved using GA on GPU.

The authors of this paper wish to inspire researcher to port computationally expensive real life problems using GAs / PGAs on GPU.

ACKNOWLEDGMENT

We express our sincere thanks to all the authors, whose papers in the area of evolutionary algorithm based on GPU are published in various conference proceedings and journals.

REFERENCES

[1] W. Withayachumnankul, B. Laksanapanai and C. Pintavirooj, “Hardware-accelerated Objective Function Evaluation for Medical Image Registration”, *IEEE Region 10th Conference on TENCON*, Vol. 1, pp. 419-422, 2004.

[2] M. Wong, T. Wong and K. Fok, “Parallel Evolutionary Algorithms on Graphics Processing Unit”, *IEEE Congress on Evolutionary Computation*, Vol. 3, pp. 2286-2293, 2005.

[3] Z. Luo and H. Liu, “Cellular Genetic Algorithms and Local Search for 3-SAT Problem on Graphic Hardware”, *IEEE Congress on Evolutionary Computation*, pp. 2988-2992, 2006.

[4] M. Wong and T. Wong, “Parallel Hybrid Genetic Algorithms on Consumer-level Graphics Hardware”, *IEEE Congress on Evolutionary Computation*, pp. 2973-2980, 2006.

[5] J. Li, X. Wang, R. He and Z. Chi, “An Efficient Fine-grained Parallel Genetic Algorithm Based on GPU-Accelerated”, *IFIP International Conference on Network and Parallel Computing - Workshops,* pp. 855-862, 2007.

[6] S. Debattisti, N. Marlat, L. Mussi and S. Cagnoni, “Implementation of a Simple Genetic Algorithm within the CUDA Architecture”, *Genetic and Evolutionary Computation Conference,* 2009.

[7] S. Tsutsui and N. Fujimoto, “Solving Quadratic Assignment Problems by Genetic Algorithms with GPU Computation: A Case Study”, *Proceedings of the 11th annual conference companion on Genetic and evolutionary computation conference,* pp. 2523–2530, 2009.

[8] L. Zaloudek, L. Sekanina and V. Simek, “GPU Accelerators for Evolvable Cellular Automata”, *Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns,* pp. 533-537, 2009.

[9] D. Nie, K. Han and H. Lee, “Stereo Matching Algorithm using Population-based Incremental Learning on GPU”, *IEEE International Workshop on Intelligent Systems and Applications,* pp. 1-4, 2009.

[10] D. Robillard, V. Marion and C. Fonlupt, “High Performance Genetic Programming on GPU”, *Proceedings of the 2009 Workshop on Bio-inspired Algorithms for Distributed Systems*, pp. 85-94, 2009.

[11] M. Wong and T. Wong, “Implementation of Parallel Genetic Algorithms on Graphics Processing Units”, *Intelligent and Evolutionary Systems, Studies in Computational Intelligence*, Vol. 187, pp. 197-216, 2009.

[12] R. Arora, R. Tulshyan and K. Deb, “Parallelization of Binary and Real-coded Genetic Algorithm on GPU using CUDA”, *IEEE Congress on Evolutionary Computation*, pp. 1-8, 2010.

[13] M. Wong and G. Cui, “Data Mining using Parallel Multi-objective Evolutionary Algorithms on Graphics Hardware”, *IEEE Congress on Evolutionary Computation*, pp. 1-8, 2010.

[14] P. Vidal and E. Alba, “A Multi-GPU Implementation of a Cellular Genetic Algorithm”, *IEEE Congress on Evolutionary Computation*, pp. 1-7, 2010.

[15] S. Kannan and R. Ganji, "Porting Autodock to CUDA", *IEEE Congress on Evolutionary Computation*, pp. 1-8, 2010.

[16] C. Salwala, V. Kotrajaras and P. Horkaew, "Improving Performance for Emergent Environments Parameter Tuning and Simulation in Games Using GPU", 3rd *IEEE International Conference on Computer Science and Information Technology*, Vol. 2, pp. 37-41, 2010.

[17] M. Yoshimi, Y. Kurano, M. Miki and T. Hiroyasu, "An Implementation and Evaluation of CUDA-based GPGPU Framework by Genetic Algorithms", *International Journal of Computer Science and Network Security*, Vol. 10, No. 12, pp. 29-37, 2010.

[18] M. Oiso, T. Yasuda, K. Ohkura and Y. Matumura, "Accelerating Steady-state Genetic Algorithms Based on CUDA Architecture", *IEEE Congress on Evolutionary Computation*, pp. 687-692, 2011.

[19] Y. Ke, Y. Li and D. Li, "Image Matching using Genetic Algorithm on GPU ", *International Conference on Control, Automation and Systems Engineering*, pp. 1-4, 2011.

[20] Y. Sato, N. Hasegawa and M. Sato, "GPU Acceleration for Sudoku Solution with Genetic Operations", *IEEE Congress on Evolutionary Computation*, pp. 296-303, 2011.

[21] A. Munawar, M. Wahib, M. Munetomo and K. Akama, "Advanced Genetic Algorithm to Solve MINLP Problems over GPU", *IEEE Congress on Evolutionary Computation*, pp. 318-325, 2011.

[22] R. Bogdanski, L. Peter and T. Becker, "Improving Scheduling Techniques in Heterogeneous Systems with Dynamic, On-Line Optimizations", *International Conference on Complex, Intelligent, and Software Intensive System*, pp. 496-501, 2011.

[23] P. Kromer, J. Platos, V. Snasel and A. Abraham, "A Comparison of Many-threaded Differential Evolution and Genetic Algorithms on CUDA", *Third World Congress on Nature and Biologically Inspired Computing*, pp. 509-514, 2011.

[24] S. Yoo, M. Harman and S. Ur, "Highly scalable multi objective test suite minimization using graphics cards", *Proceedings of the Third International Conference on Search Based Software Engineering*, pp. 219-236, 2011.

[25] O. Maitre, F. Kruger, S. Querry, N. Lachiche and P. Collet, "EASEA: specification and execution of evolutionary algorithms on GPGPU", *Soft Computing, Springer*, Vol. 16, No. 2, pp. 261-279, 2012.

[26] J. Lin, T. Igarashi, J. Mitani, M. Liao and Y. He, "A Sketching Interface for Sitting Pose Design in the Virtual Environment", *IEEE Transactions on Visualization and Computer Graphics*, Vol. 18, No. 11, pp. 1979-1991, 2012.

[27] K. Wang and Z. Shen, "A GPU-Based Parallel Genetic Algorithm for Generating Daily Activity Plans", *IEEE Transactions on Intelligent Transaction Systems*, Vol. 13, No. 3, pp. 1474-1480, 2012.

[28] S. Harding, "Genetic Programming on Graphics Processing Units Bibliography" *Soft Computing*, pp. 1-10, 2010.

[29] D.E. Goldberg, "*Genetic Algorithms in Search, Optimization and Machine Learning*", Addison-Wesley Professional, First Edition, 1989.

[30] M. Harris, "Mapping computational concepts to GPUs", Proceedings of the *ACM Special Interest Group on Computer Graphics and Interactive Techniques Conference*, Article No. 50, 2005.

[31] M. Nowostawski and R. Poli, "Parallel Genetic Algorithm Taxonomy", *Proceedings of the 3rd International Conference on Knowledge-Based Intelligent Information and Engineering Systems*, pp. 88-92, 1999.

[32] M. A. Ismail, "Parallel Genetic Algorithms (PGAs): Master Slave Paradigm Approach Using MPI", *E-Tech*, pp. 83-87, 2004.

[33] E. Cantú-Paz and D. E. Goldberg, "On the scalability of parallel genetic algorithms", *Journal of Evolutionary Computation*, Vol. 7, No. 4, pp. 429-449, 1999.

[34] G Marsaglia, "Xorshift RNGs", *Journal of Statistical Software*, Vol. 8, No. 14, pp. 1–6, 2003.

[35] S. K. Park and K. W. Miller, "Random number generators: good ones are hard to find", *Communication of the ACM*, Vol. 31, No. 10, pp. 1192–1201, 1988.

[36] M. Oiso, Y. Matsumura, T. Yasuda and K. Ohkura, "Implementing genetic algorithms to CUDA environment using data parallelization", Technical Gazette, Hrcak Portal of scientific journals of Croatia, Vol.18, No.4, 2011.

[37] "*NVIDIA CUDA C Programming Guide*", Version 4.2, 2.11.2012,http://developer.download.nvidia.com/compute/ DevZone/docs/html/C/doc/CUDA_C_Programming_Guide .pdf.