GPGPU based Dual Population Genetic Algorithm for solving Constrained Optimization Problem

A. J. Umbarkar
Department of Information Technology
Walchand College of Engineering, Sangli
Sangli, Maharashtra
anantumbarkar@rediffmail.com

P. D. Sheth
Department of MCA
Government College of Engineering, Karad
Maharashtra
pranalisheth@gmail.com

***Abstract*—**Dual Population Genetic Algorithm is a variant of Genetic Algorithm that provides additional diversity to the main population. It covers the premature convergence problem as well as the diversity problem associated with Genetic Algorithm. But also its additional population introduces large search space that increases time required to find an optimal solution. This large scale search space problem can be easily solved using consumer-level graphics cards. The solution obtained using accelerated DPGA for solving a constrained optimization problem from CEC 2006 is compared with the obtained solution using sequential algorithm. The results show speed up maintaining solution quality.

***Keywords*—** High Performance Computing (HPC), CUDA C, Dual Population Genetic Algorithm (DPGA), Constrained Optimization Problems (COPs), Function Optimization

I. Introduction

Dual Population Genetic algorithms (DPGA) are powerful, domain independent search technique that obtains solution to optimization problems. It addresses diversity as well as premature convergence problems of Genetic Algorithm (GA). DPGA uses a reserve population along with the main population. Both populations employ selection, mutation, and crossover to generate new search points in a state space. This provides additional diversity to the main population [1]. Incurred overload of additional population gives rise to increased execution time required for evolution. Also execution time of DPGA can become a limiting factor for some large computing intensive problems, because a lot of candidate solutions must be evaluated.

Graphic Processing Units (GPUs) have increasing requirements from the video game industry, while their price remained in the range of consumer market [2, 3]. They offer floating point calculation much faster than CPU and, also they can be targeted to solve general problems that can be expressed in Single Instruction Multiple Data (SIMD) format.

This paper uses Maximum Constrains Satisfaction Method (MCSM) along with DPGA to solve Constrained Optimization Problems (COPs). MCSM is a novel technique this includes two phases. The first phase tries to satisfy maximum constrains and then the second phase attempts to optimize an objective function.

Section II provides a brief literature review about evolution of DPGA, Evolutionary Algorithms for solving COPs and implementation of GAs on GPGPU. Section III describes algorithm of DPGA and code optimization. Section IV presents experimental results and discussion. Section V gives some conclusions and future scope.

This research is supported by Nvidia,Pune for GeForce® GTX™ TITAN GPGPU card and funded by Technical Quality Improvement Program, Phase -II (TEQIP-II), Government of India.

II. Literature Review

DPGA for solving COPs on GPU is an open research problem. Therefore we studied literature about evolution of DPGA. We have also surveyed how other Evolutionary Algorithms applied to solve COPs. DPGA yet not implemented on GPUs therefore we studied how GAs are implemented on GPUs using CUDA C.

Park and Ruy (2006) [4] introduced DPGA. Park and Ruy (2007) [5] proposed DPGA-ED that is an improved design-DPGA. Unlike DPGA, the reserve population of DPGA-ED evolves by itself. Park and Ruy (2007) [6] proposed a method to dynamically adjust the distance between the populations using the distance between good parents. Park and Ruy (2007) [7] exhibited DPGA2 that utilizes two reserve populations. Park and Ruy (2010) [1] experimented DPGA on various classes of problems using binary, real-valued, and order-based representations. Umbarkar and Joshi (2013) [8] compared DPGA with CUDA C GA for Multimodal Function Optimization. The results show that the performance of OpenMP GA better than SGA on the basis of execution time and speed up. Umbarkar, Joshi, Hong (2014) [9] proposed Multithreaded Parallel DPGA (MPDPGA) which outperforms serial DPGA and simple GA.

The basic and classical constrained optimization methods include penalty function method, Lagrangian method [10] and Sequential Quadratic Programming (SQP) [11]. These are local search methods which can find a local optimal solution. Recent trend is to make use of evolutionary algorithms to solve constrained optimization problems [12, 13]. Comparing with the traditional nonlinear programming approach, evolutionary algorithms need less information such as gradient (derivatives), as well as it is a global searching approach.

Arora, Tulshyan and Deb (2010) [14] proposed the parallelization of binary & real coded GA on GPU using CUDA. Kannan and Ganji (2010) [15] proposed GPU based

GA to find the optimal docking position of a ligand to a protein. Yoshimi, Kurano, Miki et al (2010) [16] proposed a framework for implementation of parallel computing on GPU by evaluating simple genetic algorithms (SGA). They showed the relationship between computational speed and execution condition. Oiso, Yasuda, Ohkura, and Matumura (2011) [17] implemented Steady state GA on GPU using CUDA for function optimization. The results of Steady state GA on GPU are 3 to 6 faster than CPU Intel corei7 (2.8 Ghz). Munawar, Wahib, Munetomo and Akama (2011) [18] proposed Adaptive Resolution GA to solve non-convex Mixed Integer Non-Linear Programming (MINLP) and non-convex Non Linear Programming (NLP) problems over GPU.

III. GPGPU based Dual Population Genetic Algorithm

DPGA starts with two randomly generated populations viz., main population and reserve population. The individuals of each population are evaluated by their own fitness functions. The evolution of each population is obtained by inbreeding between parents from the same population, crossbreeding between parents from different populations, and survival selection among the obtained offspring [1]. The methodology for applying DPGA for COPs is described in this section. A detailed pseudo code is explained in Fig.2 entitled DPGA_MCSM. MCSM is a novel technique based on Deb's rule that in the category of methods searching for feasible solutions. It states any feasible solution is preferable to any infeasible one [19].

The objective function is used as fitness function for evolution of the main population. Fitness function for the reserve population is defined in another way. An individual in the reserve population is given a high fitness value if its average distance from each of the individuals of the main population is large. Therefore the reserve population can provide the main population with additional diversity. Equation (1) describes fitness function for reserve population. Each individual of the reserve population can maintain a given distance δ from the individuals of the main population [1, 4, 5].

$$fr_{\delta}(x) = 1 - |\delta - d(M, x)| \quad (1)$$

Where,

d(M, x): average distance $(0 \le d(M, x) \le 1)$ between the main population M and individual *x* of the reserve population

δ: desired distance $(0 \le \delta \le 1)$ between two population

This paper uses Compute Unified Device Architecture (CUDA) [cuda] framework to implement DPGA on GPU as it promises best achieved results so far.The GPU is optimised to SIMD type processing and contains hardware scheduler which swiftly swaps existing threads to hide main memory latency. Because of this, a proposed model should utilize thousands of parallel threads with minimum code branching. NVidia GPU consists of multiprocessors capable to perform tasks in parallel. Threads running in these units are very lightweight. The memory attached to graphics cards is divided into two levels main memory and on-chip memory.

The main memory has a big capacity (hundreds of MB) and holds a complete set of data as well as user programs. It also acts as an entry/output point during communication with CPU. Unfortunately, big capacity is outweighed with high latency. On the other hand, the on chip memory is very fast, but has very limited size. Apart from per-thread local registers, the on-chip memory contains particularly useful per-multiprocessor shared segments.

The CUDA C DPGA algorithm exactly emulates the sequential algorithm stated in Fig.2 except for fitness calculation function, wherein small changes are introduced to get better performance. This method evaluates fitness of each individual using a defined objective function. As fitness evaluation of each individual is an independent step it can be executed in parallel.

Accelerated GA model maps GA to CUDA API with a special focus on the massive parallelism. The focus is that every individual is controlled by a single CUDA thread. We launch a thread block of 128 threads to compute fitness values of 128 individuals of the population. In our model every CUDA thread computes fitness of objective function. In this way, this step reduces the overall time required to evaluate the fitness of all individuals in the population. And thus helps significantly to reduce the amount of time required for total execution.

The local populations are stored in shared on-chip memory on particular GPU multiprocessors. As communication between CPU and GPU happens only during results exchange, this model also avoids PCI express bandwidth bottleneck which drastically chokes performance of some existing applications.

IV. Experimental Results and Discussion

This paper solves first problem from Problem Definitions and Evaluation Criteria for the Congress on Evolutionary Computation 2006 Special Session on Constrained Real-Parameter Optimization problem [41]. In this report, 24 benchmark functions are described. Guidelines for conducting experiments, statistical parameters and its formulae, performance evaluation criteria are given at the end of this report. Table I describes function g01 from CEC 2006. It describes function with its name, dimension (D), type of function, no. of linear inequality constrains (LI), no. of non-linear inequality constrains (NI), no. of linear equality constrains (LE) and no. of non-linear equality constrains (NE).

Table II exhibits the parameter settings used for experimentation. Consecutive 30 runs are calculated for each function keeping these parameter values constant.

Size of the main population as well as the reserve population is taken as 100. Crossover rate, elitism rate, mutation rate and crossbreed rate are kept identical for both the main population and the reserve population. Sequential as well as parallel algorithm uses identical parameter setting.

TABLE I .Function Description

Function	D	Type	LI	NI	LE	NE
g01	13	Quadratic	9	0	0	0

TABLE II. . PARAMETER SETTINGS

Parameter	Value	Parameter	Value
Crossover Rate	0.80	Elitism Rate	0.10
Mutation Rate	0.09	Crossbreed Rate	0.10
Main Pop. Size	128	Reserve Pop. Size	128

1) Solution Quality

Achievable solution quality and speedup of the proposed DPGA were examined using g01 function is given in table III.

Solution obtained using proposed parallel algorithm and sequential algorithm is close from each other. Table III shows Optimal Solutions Found (OSF) in sequential as well as CUDA C DPGA algorithm. For CUDA C DPGA, the results are taken on NVIDIA GeForce GTX 9400, Tesla C1060, NVIDIA GeForce 660 and NVIDIA GTX TITAN which has 16, 240, 960 and 2688 CUDA cores respectively. Results in Table III show that, CUDA C DPGA algorithm converges and produces the same and sometimes better solution as that of the Sequential DPGA algorithm. Standard Deviation (S.D.) and Standard Error of Mean (S.E.M) produce small values that show CUDA C converges.

TABLE III. OSF IN SEQUENTIAL, CUDA C DPGA

Function-g01	**OSF by CUDA C DPGA**			
OSF in Sequential Algorithm			-10.874	
Statistical Measures	***GTX 9400***	***Tesla C1060***	***GTX 660***	***GTX Titan***
	16 cores	***240 cores***	***960 cores***	***2688 cores***
Mean	-10.765	-10.23	-11.12	-10.97
S.D	0.608	0.075	0.421	0.141
S.E.M.	0.192	0.024	0.133	0.045

2) Speedup

CUDA C DPGA for COPs using MCSM experimented on different GPUs. Table IV gives time required to converge algorithm for g01 using parameter settings given in Table II.

Speed-up measures performance gain achieved by parallelizing a given application over sequential implementation. The speed-up is the ratio of sequential run time to parallel run time.

$$S = \frac{T_s}{T_p} \qquad (2)$$

Table IV gives the speed-up using GPUs for the test problem g01.

It is observed that, as the number of cores increases, speed-up increases. The corresponding graph in Fig.1 shows speedup obtained using GPUs with increasing cores for the test function g01. Copying between host and device memory may incur a performance hit due to system bus bandwidth and latency. With the effect of that we do not get speed up for GTX 9400. With asynchronous memory transfers, handled by the GPU's DMA engine this problem can be alleviated for high end cards Tesla C1060 and GTX 660.

TABLE IV . SPEED UP OBTAINED BY CUDA C DPGA ON GPUS

Sr. No.	**Sequential**	**GTX 9400**	**Tesla C1060**	**GTX 660**	**GTX Titan**
	Time (sec)	**16 cores**	**240 cores**	**960 cores**	**2688 cores**
1	37.54	39.91	31.02	23.23	12.06
2	37.73	39.90	31.25	24.00	12.07
3	37.54	38.23	31.26	23.23	12.26
4	37.53	39.91	31.26	23.20	11.98
5	36.54	39.91	31.26	23.23	12.27
6	36.60	38.76	31.26	23.22	12.28
7	37.50	39.91	31.25	23.23	11.96
8	37.54	39.90	31.26	23.30	12.27
9	37.53	39.90	31.26	23.23	11.98
10	37.58	39.91	31.26	22.23	12.27
Avg.	37.36	39.624	31.234	23.2100	12.14

TABLE IV . SPEED UP OBTAINED BY CUDA C DPGA ON GPUS

Function	**Speed Up by CUDA C DPGA**			
	GTX 9400	***Tesla C1060***	***GTX 660***	***GTX Titan***
g01	***16 cores***	***240 cores***	***960 cores***	***2688 cores***
Speed up →	0.942	1.196	1.610	3.101

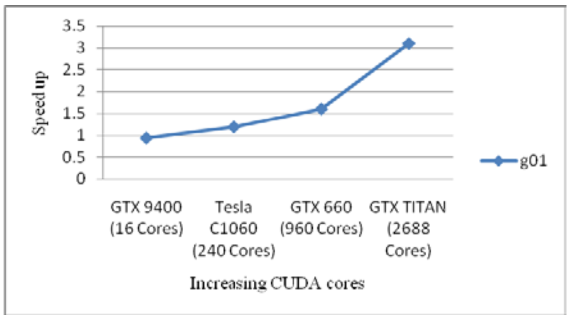

Fig.1. Speed up obtained using GPUs with increasing number of cores

Further, GPUs are targeted for computational intensive problems. Therefore, for the problems that are not sufficient complex CPU can outperform them. GPU implementation achieves better power-to-watt ratio then CPU, thus electrical energy is saved during the computation. Furthermore, the graphics card used is cheaper than any CPU running at the same speed.

V. CONCLUSION

CUDA C DPGA is a novel technique for solving COPs which aims to GPUs for general purpose programming. Experiments conduct using CEC 2006 problems set show increase in speed up with increase is no. of cores of GPUs. Copying between host and device memory may incur a performance hit. GPUs save electrical energy due to lower power-to-watt ratio compared to CPU. Furthermore, GPUs are effectively cheaper than CPUs. In future, it is expected that proposed technique will scale and give better speedup. Also using alternative constraints handling method, parallel algorithm and high performance computing paradigm a better speed up can be achieved.

ACKNOWLEDGMENT

We express our sincere thanks to all authors, whose papers in the area of DPGA, COPs and GA using CUDA are published in various conference proceedings and journals.

REFERENCES

[1] T. Park, K. Ruy, “A Dual-Population Genetic Algorithm for Adaptive Diversity Control”, IEEE transactions on evolutionary computation, vol. 14, no.6, pp 865-883, Dec. 2010, pp 865-883. DOI:10.1109/TEVC.2010.2043362

[2] NVIDIA, C.: Compute Unified Device Architecture Programming Guide. NVIDIA: Santa Clara, CA, 2007

[3] H. Nguyen. GPU Gems 3. Addison-Wesley Professional, 2007

[4] T. Park, K. Ruy, “A Dual-Population Genetic Algorithm for Balance Exploration and Exploitation”, Acta Press Computational Intelligence, 2006.

[5] T. Park and K. Ruy, “A dual population genetic algorithm with evolving diversity”, in Proc. IEEE Congress Evolutionary Computation, 2007, pp. 3516–3522. DOI:10.1109/CEC.2007.4424928

[6] T. Park and K. Ruy, “Adjusting population distance for dual-population genetic algorithm,” in Proc. Aust. Joint Conf. Artificial Intelligence, 2007, pp. 171–180. DOI:10.1109/TEVC.2010.2043362

[7] T. Park, R. Choe, K. Ruy, “Dual-population Genetic Algorithm for Nonstationary Optimization”, in Proc. GECCO’08 ACM, 2008, pp.1025-1032. DOI: 10.1145/1389095.1389286

[8] A. Umbarkar, M. Joshi, “Dual Population Genetic Algorithm (GA) versus OpenMP GA for Multimodal Function Optimization”, International Journal of Computer Applications, vol. 19, no. 64, February 2013, pp. 29-36. DOI: 10.5120/10744-5516

[9] A. Umbarkar, M. Joshi, W. Hong, “Multithreaded Parallel Dual Population Genetic Algorithm (MPDPGA) for unconstrained function optimizations on multi-core system”, Appl. Math. Comput., Vol. 243, pp. 936–949 2014,, http://dx.doi.org/10.1016/j.amc.2014.06.033

[10] D. Luenberger and Y. Ye, “Linear and nonlinear programming third edition”, New York, NY: Springer, 2007. ISBN 978-0-387-74503-9

[11] P. Boggs and J. Tolle, “Sequential quadratic programming,” Acta Numerica, vol.4, no.1, Jan. 1995, pp.1-51.

[12] C. Coello Coello, “Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art,” Comput. Methods Appl. Mech. Eng., vol.191, no.11-12, Jan. 2002, pp.1245-1287. DOI: 10.1016/j.bbr.2011.03.031.

[13] H. Lu and W. Chen, “Self-adaptive velocity particle swarm optimization for solving constrained optimization problems” J. Global Optimiz., vol.41, no.3, Jul. 2008, pp.427-445. DOI: 10.1007/s10898-007-9255-9.

[14] R. Arora, R. Tulshyan, K. Deb, “Parallelization of binary and real-coded genetic algorithm on GPU using CUDA”, IEEE explorer, 2010

[15] S. Kannan and R. Ganji, “Porting Autodock to CUDA”, IEEE explorer, 2010.

[16] Masato Yoshimi, Yuki Kurano, Mitsunori Miki et al, “An Implementation and Evaluation of CUDA-based GPGPU Framework by Genetic Algorithms”,IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.12, December 2010

[17] M. Oiso and Y. Matumura, “Accelerating Steady-state genetic algorithms based on CUDA architecture”, IEEE explorer, pp. 687-692 2011.

[18] Munawar, M. Wahib, M. Munetomo and K. Akama, “Advanced genetic algorithm to solve MINLP problems over GPU”, IEEE explorer, pp. 318-325, 2011.

[19] K. Deb, “An efficient constraint handling method for genetic algorithms, Computer Methods in Applied Mechanics and Engineering”, vol.186, no.2–4, pp. 311–338, 2000. DOI: 10.1016/S0045-7825(99)00389-8.

```
Procedure DPGA_MCSM
begin
        Initialize main population $M_0$, reserve population $R_0$, crossover rate, elitism rate, mutation rate, crossbreeding rate, tour size, max generation $t_{max}$
        Initialize $M_0$ of size $m$, accept individuals which satisfies all constrains
        Initialize $R_0$ of size $n$, $n>m$
        t: = 0
        Repeat
         Step I: Encoding of both population from decimal to binary value representation
         Step II: Fitness Calculation
           a.   Evaluate $M_0$ using objective function $fm(x)$
           b.   Evaluate $R_0$ using fitness function for reserve population (1)  $fr(x)$
        Step III: Inbreeding of main population and intermediate main population $O_m$ generation via crossover
        Step IV: Inbreeding of reserve population and intermediate reserve population $O_r$ generation via       crossover
        Step V: Crossbreeding
           a.   Offspring C of size (n-m) using best individuals from $O_m$ and $O_r$
           b.   Make $I_m = C \cup O_m$ and $I_r = C \cup O_r$
       Step VI: Decoding: Decoding of both population from binary representation to decimal representation
       Step VII: Evaluation
           a.   Evaluate $I_m$ using $fm(x)$
           b.   Evaluate $I_r$ using $fr(x)$
      Step VIII: Survival selection from $I_m$ of size m and from $I_r$ of size n
        t  = t + 1
        Until
        $fm(x) >= $ global optimal value or  $t > t_{max}$
End
Where,
t : index of current generation                    $M_0, R_0$: main, reserve population
$fm(x)$: objective function                                  $O_m, O_r$: intermediate main, reserve population respectively
$fr(x)$: fitness function for reserve population   $I_m, I_r$: constitute set of main, reserve population respectively
C: offspring                                                 $t_{max}$: maximum generations
```