





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.

$$S = \frac{T_s}{T_p} \tag{2}$$

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 Time (sec)	GTX 9400	Tesla C1060	GTX 660	GTX Titan
		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

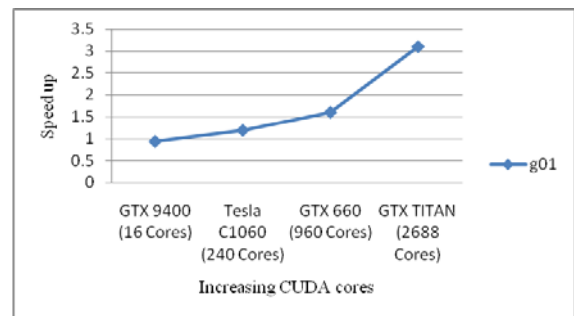


Fig.1. Speed up obtained using GPUs with increasing number of cores

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 than 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

$fm(x)$ : objective function

$fr(x)$ : fitness function for reserve population

$C$ : offspring

$M_0, R_0$ : main, reserve population

$O_m, O_r$ : intermediate main, reserve population respectively

$I_m, I_r$ : constitute set of main, reserve population respectively

$t_{max}$ : maximum generations