A Comprehensive Survey on Various Evolutionary Algorithms on GPU

Satvir Singh[1], Jaspreet Kaur[2] and Rashmi Sharan Sinha[3]
[1,2,]*Department of Electronics & Comm. Engineering,*
SBS State Technical Campus, Moga Road Ferozepur–152004, Punjab
E-mail: [1]drsatvir.in@gmail.com, [2]er.jaspreetkaur1@hotmail.com,
[3]sinharashmisinha@hotmail.com

***Abstract*—This paper presents a comprehensive survey on parallelizing computations involved in optimization problem on Graphics Processing Unit (GPU) using CUDA (Compute Unified Design Architecture). GPU have multithread cores with high memory bandwidth which allow for greater ease of use and also more radially support a layer body of applications. Many researchers have reported significant speedups with General Purpose computing on GPU (GPGPU). Stochastic meta-heuristic search algorithms, e.g., Mixed Integer Non-Linear Programming (MINLP), Central Force Optimization(CFO), Genetic Algorithms (GA), and Particle Swarm Optimization(PSO), etc. are being investigated nowadays for improved performance with processing power of GPU. From study it is found that GPGPU shows tremendous speedups from 7 times in Steady State GAs to 10, 000 times speedups in CFO.**

Keywords: GPU, GPGPU, CUDA, MINLP, PGMOEA, CGA, CFO, Optimization Algorithms

I. INTRODUCTION

General Purpose GPU Computing really took off when CUDA and Stream arrived in late 2006 [1]. GPU constitutea tremendous step towards a usable, suitable, scalable and manageable future-proof programming model [2]. Optimization works are significantly parallel, and so GPUs evolved as large-scale general purpose computation machines [3] [4] [5] [6]. With the advent and large availability of General Purpose Graphics Processing Units and the development and straightforward applicability of the Compute Unified Device Architecture platform, several applications are being benefited by the reduction of the computing time [7]. GPGPU-based architecture, aiming at improving the performance of computationally demanding optimizations for identifiable specific mapping parameters, one can reduce total execution time drastically and also, improve greatly the optimization process convergence. Application performance can be significantly improved by applying memory access pattern-aware optimizations that can exploit knowledge of the characteristics of each access pattern [3]. To evaluate the effectiveness of our methodology, we have created a tool that incorporates our proposed algorithmic optimizations and report on execution speedup using selected benchmark kernels that cover a wide range of memory access patterns commonly found in GPGPU workloads [8]. Graphics Processing Units (GPUs) are widely used among developers and researchers as accelerators for applications outside the domain of traditional computer graphics. In particular, GPUs have become a viable parallel accelerator for scientific computing with low investment in the necessary hardware.

In this paper, various sections describe different Evolutionary Algorithms (EAs) and their gained efficient speedup on GPU using CUDA. Sections II to V are dedicated to various GAs variants those have been investigated by various researchers. Sections VI to X presents Cellular automata, CFP, Multi-objective Optimization, PSO Differential Evolution (DE).Finally Section XI and XII present study on GAs Versus DE and conclusions at the end.

II. ISLAND BASED GENETIC ALGORITHM

Island based genetic algorithm (GA) is implemented on Multi-GPU in [27], for solving the Knapsack problem. The main motiveis to speed up the GA by using a cluster of NVIDIA GPU and comparing the execution time of single GPU with a multicore CPU.

The population of proposed GA is organized in two one-dimensional arrays. First array representing the genotype and the other array represents the fitness value. The GA parameter such as population and chromosome size, the crossover and mutation rates, the statistic collection and migration interval, the total number of evaluated generations etc. are filled with command line parameters, this maintained structure is stored at GPU constant memory [28]. The basic concept to maximize GPU utilization is to control thread divergence and amalgamate all memory accesses using this algorithm [28]. Firstly, a hash function generator based stateless random number is generated [29][30]. Then, the genetic material is exchanged of two parents using crossover and mutation process by performing the binary tournament selection to create a new individual. As the new offspring is created fitness evaluation is carried out. Next, the parent is replaced by the offspring with the help of entire warp if the fitness value of latter is higher than the former. The good individuals are migrated from the adjacent lower index island to the higher index island which is arranged in the unidirectional ring topology. Lastly, all the statistical data from the local island and from the global gathering process are collected [31].

The analysis illustrates that as the individual per GPU and number of islands increases the fitness value increases. Secondly, the execution time is invariant for

island size up to 512 and then elevate linearly beyond 512. All in all, the implemented Island based GA leads to the GPU performance of 5.67 TFLOPS.

III. ADVANCED GENETIC ALGORITHM

With the increasing advent of GPGPU using CUDA, the stochastic algorithm of advanced Genetic Algorithm is used to solve non-convex MINLP and non-convex Non-linear Programming (NLP) problems [9]. MINLP refers to mathematical programming algorithms that can optimize both continuous and integer variables, in a context of nonlinearities in the objective function and/or constraints. MINLP problems involve the simultaneous optimization of discrete and continuous variables. These problems often arise where one is trying to simultaneously optimize the system structure and parameters. This is difficult because optimal topology is dependent upon parameter levels and vice versa [9].

In many design optimization problems, the structural topology influences the optimal parameter settings so a simple de-coupling approach does not work: it is often not possible to isolate these and optimize each separately. Finally, the complexity of these problems depends upon the form of the objective function. In the past, solution techniques typically depended upon objective functions that were single-attribute and linear (i.e., minimize cost). However, real problems often require multi-attribute objectives such as minimizing costs while maximizing safety and/or reliability, ensuring feasibility, and meeting scheduled deadlines. In these cases, the goal is to optimize over a set of performance indices which may be combined in a nonlinear objective function. For efficient utilization of GPU parallel resources adaptive resolution genetic algorithm (arGA) is developed [9].Through this algorithm the intensity of each individual is beamed using entropy measures. The algorithm is tested for different benchmarking problems [9] having different levels of difficulty. Parallelization of arGA and the arLS (local search) operators is done to gaina significant speedup. The results of the tests shows a speedup of 42x with single precision and 20x with double precision overnVidia Fermi C2050 GPU [9].

IV. STEADY STATE GENETIC ALGORITHM

Steady-State GA is implemented on GPU using CUDA in [23], where population individual data is accessed parallel to effectively speedup the process. The optimization problem is effectively solved by the means of Evolutionary Computing [24]. The steady state Genetic Algorithm is used to access optimization algorithms. These algorithms basically have selection for the reproduction and selection of survival implementation with concurrent kernel execution [22].

The implementation of Steady-State GA is done as follows: Firstly, from the population two individual (parents) are received by Streaming Multiprocessors (SM). Then, the kernel generates random number as GAs are stochastic search processes. BLX-α is adopted as blend crossover in the crossover process. The crossover operation is executed with two parents in SM, and the two offspring yielded are stored in the shared memory. Next, uniform mutation, fitness based sorting process and selection process is executed. This whole process is repeated until the loop terminates. Four test functions of the optimization problem Hyper sphere, Rosen rock, Ackley, and Griewank were used for comparing GPU and CPU computation on implementing Steady-State GA. The study is first performed upon CPU then with nVidia GeForce GTX480GPU gives a speedup of 3x to 6x then the previous implementation on CPU [19]. Moreover, the speed up ratio for Generational GA is much better than Steady-State GA on GPU since computational granularity is very small in the latter. So a large amount of execution time is occupied by the latency caused by the kernel calls. However, in terms of function values Steady-State GA are more efficient.

V. CELLULAR GENETIC ALGORITHM

Genetic Algorithm have a subclass known as Cellular Genetic Algorithm (cGA) which provides the data of population structured in several specified topologies [19]. Cellular Genetic Algorithm (cGA) is implemented for multi-GPU to accelerate the execution process so that the system could be more efficient. cGAis used because of its high performance and swarm intelligence structure. Until the global optimum region is reached, cGA is able of keeping a high diversity in population.

To manage the multi-GPU utilization each CPU thread is held responsible for one GPU device which is known as multi-threaded mode. Firstly, each CPU thread is associated to one GPU. This can be done if common structure (toroidal grid) is designed for all CPU threads. Then, the population is divided into subpopulations which is stored in the global memory of each GPU. Each GPU works individually irrespective of other GPU and the process is same as performed with single GPU implementation. To ensure that every GPU had finished its work a synchronization barrier is used and lastly, data is collected and transferred to other GPUs. The process executes until the while loop in pseudo code of cGA terminates.

Three discrete optimization problems: Colville Minimization, Error Correcting Codes Design Problem (ECC) and Massively Multimodal Deceptive Problem (MMDP), and three continuous ones, Shifted Griewank function, Shifted Restringing function and Shifted Rosen rock function [20] [21] were selected for comparing the algorithm in terms of efficiency and efficacy. Statistical tests [19] are performed for each problem to ensure that the results are statistically significant. A common

parameter, population size is used to make a meaningful comparison among all the algorithms.

The analysis shows the average speed up with respect to CPU version ranges from 8 to 771 and for single GPU it is alike multi-GPU, with a little overhead in the latter case. The multi-GPUis more prominent in paralleling the algorithm and producing accurate results as there is a need of special maintenance to perform same experiment upon single GPU.

Genetic Algorithm is tested and evaluated on parallel implementation on C-CUDA API on the parameters like population size, number of threads, problem size and problem of differing complexities with variation in the population individuals [19]. For an efficient implementation on GPGPU the solution is thoroughly implemented along with the operators like random number generation, initialization, selection operation, and mutation operations [13]. The nVidia GeForce 8800GTX shows overall speedup of 40–400 on three different test problems [22]. Thus parallel implementation is more effective then sequential process as compared with clock time and accuracy.

VI. Cellular Automata

Cellular Automata have various real life applications like physical system modeling, road traffic simulation, artificial life simulation, etc. [14] [15] [16]. Cellular automata design evolved from evolutionary algorithm and a part of Genetic Algorithm which is complex in nature.

The Algorithm is parallelized and implemented upon GPGPU shows an efficient reduction in execution time. The rules of Cellular Automata take longer time period in evolution in sequential execution. The same Genetic Algorithm shows 31.34x to 314.94x speedup when executed upon nVidia GeForce FX280 GPU which is a significant reduction in execution time [17].

VII. Central Force Optimization

The metaheurestic algorithm CFO is implemented upon GPGPU using local neighborhood and implemented CFO concepts [25]. The calculation of CFO independent upon the movement of probes which are scattered allover the space. The probes then slowly move towards the probehaving highest mass or fitness. PR-CFO is the most evaluated algorithm with the measures of initial position and acceleration vectors, fitness evaluation and probe movements [26]. The test problems are having the dimension of 30 to 100 of four different examples of Pseudo random CFO (PR-CFO). The PR-CFO is tested with four test types i.e. Ring, Standard, CUDA, CUDARing. PR-CFO shows a speedup of 4 to 400 using CUDA. PRCFOring and PR-CFO CUDA ring on nVidia Tesla C1060 shows10, 000 times faster results as compared with standard PR-CFOalgorithm [26].

VIII. PGMOEA

The general Purpose GPU is efficiently used in optimizing the multiple objective problems. The particle gradient Multi-objective Evolutionary Algorithm (PGMOEA) is used to solve optimization problems. PGMOEA is first experimented on CPU and then after parallelizing the algorithm executed upon GPU which formed agreat speedup results [18]. The first step to implement PGMOEA is to read parameters such as population size, dimension size, maximum iterative generations, crossover rate, mutation rate and initialize particle texture array. Blank texture array i.e objective, rank value, entropy and free energy array are then generated to store different results. Next, the particle texture array is is loaded to GPU to calculate the rank of all the particles and the results are then stored in rank value texture array. The particles are sorted in the decreasing order of their ranks to make a mating pool. The higher order rank value particles are selected to perform crossover and mutation operation using Guo's algorithm. The new particles generated through this process are then replaced with the last particles which have lower rank in mating pool to get a new population. The program is terminated if the halt condition is satisfied else particle texture array is again loaded to GPU and the process is repeated again.

TABLE I SPEEDUP COMPARISON (SOURCE [18])

PGMOE Algorithm	Example 1		Example 2	
	Time (s)	Speedup	Time (s)	Speedup
On GPU	0.97	9.95	0.83	10.64
On CPU	9.01	1.04	8.02	1.10

The experiment is conducted upon two different examples. The first example shows a speedup of 9x with nVidia GeForceGTX285 then CPU result, while the second example is 10x faster than that of CPU [18]. The speedup comparison is shown bellowing Table I.

IX. PSO Algorithm

PSO is a met heuristic algorithm works by having a swarm of particles [10]. These particles are moved around in the search-space according to a few simple formulae. The movements of the particles are guided by their own best known position in the search-space as well as the entire swarm best known position [11].

When improved positions are being discovered these will then come to guide the movements of the swarm. PSO is one of the types of Evolutionary Algorithm used to optimize the multiple objective problems. When an optimization problem involves more than one objective function, the task of finding one or more optimal solutions is known asmulti-objective optimization [10].

For implementing PSO code in C-CUDA the allocation of vector/matrix is done on the device.

Random numbers are generated using Mersenn Twister code and then based on objective functions are evaluated. After evaluation the global best particle of whole swarm is updated. Next, the sum and multiplication operations are performed on the vectors which describe the particle. The benchmark functions with many local minima mentioned in table A1 appendix [12] are selected. The algorithm is tested upon three different platforms of C, Matlab and C-CUDA. The parallel implementation of PSO on nVidia GTX 280 gives 17 to 41 times speedup in computing time in C-CUDA as compared with the Cand Matlab as Shown in Fig. 1 [12].

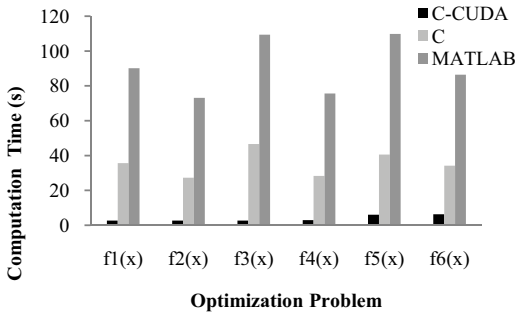

Fig. 1 Computing Time for C-CUDA, C, and MATLAB (Source [12])

TABLE II RUNNING TIME RESULTS USING C-CUDA AND C FOR THE BENCHMARK OPTIMIZATION PROBLEMS WITH 100 DIMENSIONS, 100 INDIVIDUALS, 10, 000 AND 100, 000 ITERATIONS (SOURCE [13])

Benchmark Functions	Implementation Language	10, 000 Iterations			100, 000 Iterations		
		Computing Time(s)	Standard Deviation	Speedup	Computing Time(s)	Standard Deviation	Speedup
Schwefel-F_1(x)	C-CUDA	0.64	0.01	NA	27.72	0.45	NA
	C	9.59	0.21	15.05	983.65	30.91	35.48
Rastrigin-F_2(x)	C-CUDA	0.64	0.01	NA	27.47	0.38	NA
	C	8.39	0.27	13.15	900.97	29.50	32.80
Ackley-F_3(x)	C-CUDA	0.72	0.01	NA	36.33	1.38	NA
	C	7.10	0.25	9.91	736.68	39.56	20.28
Griewank- F_4(x)	C-CUDA	0.69	0.01	NA	31.44	0.37	NA
	C	10.39	0.44	14.96	1005.35	4.33	31.98
Generalized penalized function-F_5(x)	C-CUDA	0.76	0.02	NA	37.88	1.16	NA
	C	13.75	1.13	18.07	1344.59	72.33	35.50
Generalized penalized function-F_6(x)	C-CUDA	0.72	0.01	NA	37.44	1.50	NA
	C	13.76	1.36	19.04	1300.11	81.40	34.73

TABLE III COMPARISON TABLE OF DIFFERENT EVOLUTIONARY ALGORITHMS ON GPU AND CPU

Algorithm		Experimental Set up		Time		Speed up	
		GPU (nVidia)	CPU	GPU	CPU	GPU	CPU
Island based GA		GTX580	Intel Xeon Six-Core	5.67 TFLOPS	–	653.68	11.32
Advanced GA		C2050 (Double precision)	Intel Core 2 Duo	–	–	20x	
		C2050 (Single precision)	Intel Corei7			40x	
Steady-state GA		Geforce GTX480	Intel Core i7	4.874 - 4.780s	14.46-28.56s	3.0x-6.0x	
Cellular GA		GTX-285	Intel Quad processor	0.021 - 1.821s	0.266 - 1450.415s	8 - 771	–
Binary and Real coded GA		Tesla C1060	AMD Athlon 64 X2 Dual Core	RGA 0.003365s - 22.534169s	RGA 0.071639 - 4851.69		RGA 21.289 - 215.304
Cellular Automata	Laptop	GeForce 8600M GS	Intel Core 2 Duo Processor	–	–	31.34x	
	Work-station	GeForce FX 280	Intel Core 2 Duo Processor	–	–	314.97x	
PGMEOA		GeForce GTX285	Intel Core (TM)2 Q8200	0.83s - 0.97s	8.02s - 9.01s	9.95 - 10.64	1.04 - 1.10
CFO		Tesla C1060	Intel Xeon 5504 Quad Core	–	–	10, 000x	

Table III (Contd.)…

…Table III (Comparison Table of Different Evolutionary Algorithms on GPU and CPU)

Algorithm	Experimental Set up		Time				Speed up	
	GPU (nVidia)	CPU	GPU		CPU		GPU	CPU
PSO	GTX 280	AMD Athlon x2 3800 + 2.0 GHz Dual Core	–		–		17 to 41x	
DE	GTX285	AMD Athlon x2 5200 + 2.7 GHz Dual Core	$F_1(x)$ 5.84	$F_6(x)$7.2 5	$F_1(x)$481. 38	$F_6(x)$682. 76	20x to 35x	
Many threaded DE and GA	Tesla C2050	Dual Core AMD Opteron	–		–		DE 19 - 34 x	

X. Differential Evolutionary Algorithm

GPGPU is proved to be great architectural unit in reducing the processing time [13]. The Differential Algorithm which is one of the parts of EAs is implemented upon CPU using C-CUDA. The motivating features of Differential Algorithm are easy for parallelization and convergence properties which intern gives an appropriate result. The algorithm is first tested upon CPU then on nVidia GTX285 with 1GB GDDR3 GPU with the speedup outcomes.

The implementations of DE algorithm and benchmark functions are same as used for PSO implementation on GPU. GPU gives 20x to 35x faster results which proves GPU is much more effective and efficient than Differential Algorithm on CPU [13]. The Speedup comparison results are shown in Table II.

XI. DE Versus GAs

In this paper [32], two evolutionary meta-heuristic algorithms DE and GAs, many threaded implementation is done on CUDA and results were compared when Independent task scheduling is solved. Mapping of set of task to a set of resources is known as Independent task scheduling [33] [34]. Since it is a NP-complete problem so two objectives make span and flow time are used during task mapping for optimization.

Whole meta task can be accelerated by minimizing make span and the efficient utilization of the computing environment can be done by minimizing flow time.

$$makespan = \min_{S \epsilon Sched}\{\max_{j \epsilon Jobs} F_j\} \quad (1)$$

$$flowtime = \min_{S \epsilon Sched}\{\sum_{j \epsilon Jobs} F_j\} \quad (2)$$

Real coordinates are used in DE [35]for encoding real vectors. Truncation of real encoded vector coordinates is done to translateit into schedule representation. For this fitness function f(S):Sched→R is defined which evaluates each schedule

$$f(S) = \lambda makespan(S) + (1 - \lambda)\frac{flowtime(S)}{m} \quad (3)$$

For minimization purpose a standard proposed in [34] is used. The simulation matrix is derived from ETC matrix. The time taken by the machine to execute a task can be estimated using Expected Time to Compute (ETC) matrix. The average final fitness value is calculated by both the algorithms for each ETCmatrix. The analysis shows that the DE is better than GA for solving Independent task scheduling problem and leads to better results for many threaded implementation on CUDA.

XII. Conclusion

In this paper we present different optimization algorithm with tremendous speedups in the computation time. The overall GPU performance of multi-GPU Island-based GA for solving Knapsack problem reaches 5.67 TFLOPS. MINLP archived an overall speedup of 20x to 42x using nVidia Tesla C2050 GPU as compared to Intel Core i7 920 CPU processor. On implementing Steady state GA on a GPU approximately 6 times faster results are obtained than the corresponding CPU implementation. The implementation of Cellular Genetic Algorithm for a multi-GPU platform leads to speedup range from 8 to 771 with respect to the CPU version. The new binary-coded and real-coded Genetic Algorithm using CUDA leads to a performance improvement with the speedup of 40x to 400x. Central Force Optimization (CFO)results in reduction of computing time and a speedup of 10, 000x. The computing time gets accelerated up to 17 to 41 times in C-CUD Aafter PSO is paralleled implemented on NVIDIA GTX280. GPU is much more effective and efficient for Differential Evolutionary algorithm since it gives 20x to 35x faster results than CPU. The Cellular Automata shows 314.97x as compared with the sequential implements. The advantage of GPU computing is that it is fast and cheap. A theoretical 1.5 TFLOPS is obtained by the newest nVIDIA GTX 580 at $500. The major drawback is not all algorithms can have theoretical speedup and are hard to program.

References

[1] K. S. Perumalla, “Discrete-event Execution Alternatives on General Purpose Graphical Processing Units (GPGPU), ” in *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*, 2006, pp. 74–81.

[2] J. A. Jablin, P. McCormick, and M. Herlihy, “Scout: high-performanceheterogeneous computing made simple, ” in *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011*, 2011, pp. 2093–2096.

[3] Bustamam, K. Burrage, and N. A. Hamilton, "Fast Parallel Markov Clustering in Bioinformatics using Massively Parallel Graphics Processing Unit Computing, " in *2010 Ninth International Workshop on Parallel and Distributed Methods in Verification and Second International Workshop on High Performance Computational Systems Biology*, 2010, pp. 116–125.

[4] C. Xue-bin *et al.*, "Data Processing in Space Weather Physics Models in the Meridian Project, " in *2010 Ninth International Symposium on Distributed Computing and Applications to Business Engineering and Science (DCABES)*, 2010, pp. 342–345.

[5] F. Pei-qin, D. Liang-Long, L. Xiao-Ting, and J. Chao-bo, "Design and Implementation of Remote Parallel Computing System based on Multi-Platform, " in *2010 International Conference on Internet Technology and Applications*, 2010, pp. 1–4.

[6] M. Al Hajj Hassan and M. Bamha, "An Efficient Parallel Algorithm for Evaluating Join Queries on Heterogeneous Distributed Systems, " in *2009 International Conference on High Performance Computing (HiPC),* IEEE, 2009, pp. 350–358.

[7] D. Luebke, M. Harris, N. Govindaraju, A. Lefohn, M. Houston, J. Owens, M. Segal, M. Papakipos, and I. Buck, "GPGPU: GeneralPurpose Computation on Graphics Hardware, " in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, 2006, p.208.

[8] W. B. Langdon and M. Harman, "Evolving a CUDA Kernel from an nVidiaTemplate, " in *2010 IEEE Congress on Evolutionary Computation (CEC)*, 2010, pp. 1–8.

[9] Munawar, M. Wahib, M. Munetomo, and K. Akama, "Advanced Genetic Algorithm to Solve MINLPProblems over GPU, " in*2011 IEEE Congress on Evolutionary Computation (CEC)*, 2011, pp. 318–325.

[10] J. Kennedy, J. F. Kennedy, and R. C. Eberhart, *Swarm Intelligence.* Morgan Kaufmann, 2001.

[11] J. Kennedy and R. Mendes, "Population Structure and Particle Swarm Performance, " *Proceedings of the World on Congress on Computational Intelligence*, vol.2, pp. 1671–1676, 2002.

[12] L. de P Veronese and R. A. Krohling, "Swarm's Flight: Accelerating the Particles using c-CUDA, " in *IEEE Congress onEvolutionary Computation*, 2009, pp. 3264–3270.

[13] L. De Veronese and R. A. Krohling, "Differential Evolution Algorithm on the GPU with c-CUDA, " in *2010 IEEE Congress onEvolutionary Computation (CEC)*, 2010, pp. 1–7.

[14] L. J. Durbeck and N. J. Macias, "The Cell Matrix: An Architecture for Nanocomputing, " *Nanotechnology*, vol. 12, no. 3, p. 217, 2001.

[15] M. Gardner, "Mathematical Games: The Fantastic Combinations of John ConwaysNew Solitaire Game Life, " *Scientific American*, vol. 223, no.4, pp. 120–123, 1970.

[16] M. Tomassini, M. Sipper, and M. Perrenoud, "On the Generation of High-quality Random Numbers by Two-dimensional Cellular Automata, "*IEEE Transactions onComputers,* vol. 49, no. 10, pp. 1146–1151, 2000.

[17] L. Zaloudek, L. Sekanina, and V. Simek, "GPU Accelerators for Evolvable Cellular Automata, " in *2009Computation World:Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*, 2009, pp. 533–537.

[18] X. Yue, Z. Wu, and K. Li, "Particle Gradient Multi-Objective Evolutionary Algorithm based on GPU with CUDA, " in *2010 International Symposium onInformation Science and Engineering (ISISE)*, 2010, pp. 540–544.

[19] P. Vidal and E. Alba, "A Multi-GPU Implementation of a CellularGenetic Algorithm, " in *2010 IEEE Congress onEvolutionary Computation (CEC)*, 2010, pp. 1–7.

[20] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization, " *KanGAL Report*, vol. 2005005, 2005.

[21] K. Tang, X. Y´ao, P. N. Suganthan, C. MacNish, Y.-P. Chen, C.-M. Chen, and Z. Yang, "Benchmark Functions for the CEC 2008 Special Session and Competition on Large Scale Global Optimization, " *Nature Inspired Computation and Applications Laboratory*, *USTC, China*, 2007.

[22] R. Arora, R. Tulshyan, and K. Deb, "Parallelization of Binary and Real-Coded Genetic Algorithms on GPU using CUDA, " in *2010 IEEE Congress onEvolutionary Computation (CEC)*, 2010, pp. 1–8.

[23] M. Oiso, T. Yasuda, K. Ohkura, and Y. Matumura, "Accelerating Steady-State Genetic Algorithms based on CUDA Architecture, " in *2011 IEEE Congress onEvolutionary Computation (CEC)*, 2011, pp. 687–692.

[24] F. Stentiford, "An Evolutionary Programming Approach to the Simulation of Visual Attention, " in *Proceedings of the 2001 Congress onEvolutionary Computation,* vol. 2, 2001, pp. 851– 858.

[25] Stefek, "Benchmarking of Heuristic Optimization Methods, " in 14th *International SymposiumMECHATRONIKA*, 2011, pp. 68–71.

[26] R. Green, L. Wang, M. Alam, and R. A. Formato, "Central Force Optimization on a GPU: A Case Study in High Performance Metaheuristics using Multiple Topologies, " in *2011 IEEE Congress onEvolutionary Computation (CEC)*, 2011, pp. 550–557.

[27] J. Jaros, "Multi-GPUIsland-based Genetic Algorithm for Solving the Knapsack Problem, " in *2012 IEEE Congress onEvolutionary Computation (CEC)*, 2012, pp. 1–8.

[28] C. NVIDIA, "CUDACBest Practices Guide ver. 4.0, " 2011.

[29] C. Toolkit, "4.0 CURANDGuide. nVidiaCorporation, version 12.3, January 2011."

[30] J. K. Salmon, M. A. Moraes, R. O. Dror, and D. E. Shaw, "Parallel Random Numbers: As Easy as 1, 2, 3, " in*2011 International Conference forHigh Performance Computing, Networking, Storage and Analysis (SC)*, 2011, pp. 1–12.

[31] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 2010.

[32] P. Kromer, J. Platos, V. Snasel, and A. Abraham, "A Comparison of Many-Threaded Differential Evolution and Genetic Algorithms on CUDA, " in *2011 Third World Congress onNature and Biologically Inspired Computing (NaBIC)*, 2011, pp. 509–514.

[33] S. Ali, T. D. Braun, H. J. Siegel, and A. A. Maciejewski, "HeterogeneousComputing, " 2002.

[34] T. D. Braun, H. J. Siegel, N. Beck, L.L. B¨ol¨oni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen *et al.*, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems*, " Journal of Parallel and Distributed computing,* vol. 61, no. 6, pp. 810–837, 2001.

[35] K. Price, R.M. Storn, and J.A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization.* Springer, 2006.