A Comprehensive Survey on Various Evolutionary Algorithms on GPU

Satvir Singh¹, Jaspreet Kaur² and Rashmi Sharan Sinha³

¹²Department of Electronics & Comm. Engineering, SBS State Technical Campus, Moga Road Ferozepur–152004, Punjab
E-mail: ¹drsatvir.in@gmail.com, ²er.jaspreetkaur1@hotmail.com, ³sinharashmisinha@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). GPUs 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 motive is 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 to the GPU performance of 5.67 TFLOPS.

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-a 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 GTX480 GPU 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 Restrstring 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
The metaheuristic 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 all over the space. The probes then slowly move towards the probebehaving 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, CUDA Ring. PR-CFO shows a speedup of 4 to 400 using CUDA. PR-CFORing and PR-CFO CUDA ring on nVidia Tesla C1060 shows 10,000 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 a great 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>
<thead>
<tr>
<th>Algorithm</th>
<th>Example 1</th>
<th>Example 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGMOEA On GPU</td>
<td>0.97</td>
<td>9.95</td>
</tr>
<tr>
<td>On CPU</td>
<td>9.01</td>
<td>1.04</td>
</tr>
</tbody>
</table>

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 as multi-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 C and Matlab as shown in Fig. 1 [12].

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

<table>
<thead>
<tr>
<th>Benchmark Functions</th>
<th>Implementation Language</th>
<th>10,000 Iterations</th>
<th>100,000 Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Computing Time(s)</td>
<td>Standard Deviation</td>
</tr>
<tr>
<td>Schwefel- F_1(x)</td>
<td>C-CUDA</td>
<td>0.64</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>9.59</td>
<td>0.21</td>
</tr>
<tr>
<td>Rastrigin- F_2(x)</td>
<td>C-CUDA</td>
<td>0.64</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>8.39</td>
<td>0.27</td>
</tr>
<tr>
<td>Ackley-F_3(x)</td>
<td>C-CUDA</td>
<td>0.72</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>7.10</td>
<td>0.25</td>
</tr>
<tr>
<td>Griewank- F_4(x)</td>
<td>C-CUDA</td>
<td>0.69</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>10.39</td>
<td>0.44</td>
</tr>
<tr>
<td>Generalized penalized function-F_5(x)</td>
<td>C-CUDA</td>
<td>0.76</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>13.75</td>
<td>1.13</td>
</tr>
<tr>
<td>Generalized penalized function-F_6(x)</td>
<td>C-CUDA</td>
<td>0.72</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>13.76</td>
<td>1.36</td>
</tr>
</tbody>
</table>

**Table III Comparison Table of Different Evolutionary Algorithms on GPU and CPU**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Experimental Set up</th>
<th>Time</th>
<th>Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Island based GA</td>
<td>GPU (nVidia) GTX580</td>
<td>5.67 TFLOPS</td>
<td>653.68</td>
</tr>
<tr>
<td>Advanced GA</td>
<td>CPU Intel Xeon Six-Core</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Steady-state GA</td>
<td>GPU C2050 (Double precision)</td>
<td>4.874 - 4.780s</td>
<td>14.46-28.56s</td>
</tr>
<tr>
<td>Cellular GA</td>
<td>GPU GTX-285</td>
<td>0.021 - 1.821s</td>
<td>0.266 - 1450.415s</td>
</tr>
<tr>
<td>Binary and Real coded GA</td>
<td>GPU Tesla C1060</td>
<td>AMD Athlon 64 X2 Dual Core</td>
<td>RGA 0.003365s - 22.534169s</td>
</tr>
<tr>
<td>PGMEOA</td>
<td>GPU GeForce GTX285</td>
<td>AMD Core (TM)2 Q8200</td>
<td>0.83s - 0.97s</td>
</tr>
<tr>
<td>CFO</td>
<td>GPU Tesla C1060</td>
<td>Intel Xeon 5504 Quad Core</td>
<td>–</td>
</tr>
</tbody>
</table>

Table III (Contd.)
A Comprehensive Survey on Various Evolutionary Algorithms on GPU

X. DIFFERENTIAL EVOLUTIONARY ALGORITHM

GPGPU is proved to be a 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.

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

\text{flowtime} = \min_{S\text{Sched}} \{\sum_{j\text{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 translate it into schedule representation. For this fitness function \( f(S): \text{Sched} \rightarrow \mathbb{R} \) is defined which evaluates each schedule

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

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 ETC matrix.

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 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-CUDA after 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 implemenets. 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


