Running Compact Genetic Algorithm on Large Scale Problems Using Graphics Processing Unit

Peera Thontirawong, Alongkot Burutarchanai, Sunisa Rimcharoen and Prabhas Chongstitvatana
Department of Computer Engineering, Chulalongkorn University
Bangkok, Thailand
{peera.t, alongkot.b, sunisa.r}@student.chula.ac.th, prabhas@chula.ac.th

Abstract

Since large-scale global optimization problems have a very large search space, most algorithms are not efficient to find an acceptable solution in a given execution time. Therefore, this paper proposes a performance enhancement of the Compact Genetic Algorithm, which is a simple evolutionary algorithm, by using a Graphics Processing Unit. The enhanced Compact Genetic Algorithm is competent for solving the large-scale global optimization problems because of its fast computation time and small memory requirement. The experimental results on large-scale global optimization problems show that this approach is able to find acceptable solutions within a reasonable time. We achieved the average performance speedup of 34.45 on 50,000-dimension problems, and the speedup is increased for larger size problem.

Keywords: GPU, Large scale problem, compact GA.

1. Introduction

A large-scale global optimization (LSGO) problem is a hard problem of finding the optimal solution from numerous possible solutions; hence, ordinary approximation algorithms are not suitable for these problems because the amount of memory and time consumed. Therefore, a specially designed algorithm is required to solve these problems.

The Compact Genetic Algorithm (cGA) [1] is a simple evolutionary algorithm suitable for optimization problems. The cGA uses a vector of probabilities to represent the whole population; thus, it demands little memory space. In every generation, two individuals are generated from this vector, and each probability in this vector is independently updated toward the stronger individual. The cGA mimics the Simple Genetic Algorithm (sGA) with uniform crossover behavior, while using lesser memory. Therefore, the cGA achieves comparable solution quality as the sGA with approximately the same number of fitness evaluations. Moreover, the minimal hardware requirement makes cGA desirable for hardware implementation [2].

The Graphics Processing Unit (GPU) is a specialize hardware for 3D graphics rendering. The GPU contains many small processors operated concurrently; thus, it can be viewed as low cost parallel processors. Due to the advantage of the GPU computation power, its applications in scientific computing became popular [3]. Recently, the GPU is widely used in many domains included in the field of evolutionary computing [4].

This paper presents a parallel programming of the GPU in a different way. For the evolutionary algorithm that has small size of candidate solutions, such as the cGA, parallelization by number of candidate solutions does not utilize the computation power of the GPU effectively. Hence, we focus on parallelization by the problem size instead. Since each dimension in the probability vector of cGA is independent; thus, the cGA can be performed in parallel on GPU. Therefore, the performance is sufficient to solve the LSGO problems.

The rest of this paper is organized as follows. Section 2 is an overview of the cGA. Section 3 describes the implementation, and the experiments and results are presented in section 4. Finally, this paper is concluded in section 5.

2. The Compact Genetic Algorithm

The Compact Genetic Algorithm (cGA) proposed by Harik, Lobo and Goldberg [1] represents the population as a probability distribution over the set of solutions; thus, the whole population needs not to be stored. At each generation, cGA samples individuals according to the probabilities specified in the probability vector. The individuals are evaluated and the probability vector is updated towards the better individual. The cGA mimics the order-one behavior of Simple Genetic Algorithm (sGA) with uniform crossover using a small amount of memory, and achieves comparable quality with approximately the same number of fitness evaluations as the sGA. The process of the cGA is shown in figure 1.

In the first step, the probability vector is initialized with 0.5. Each dimension in the vector represents the probability of each bit happened to be one. Two candidate solutions are sampled from this vector. After evaluating, the winner and loser are specified. From figure 1, the winner is 11100101 and the loser is 10001100. The probability vector is updated according to the winner. The different bit between the winner and loser guides the probability to come closer to the better

1) Initialize probability vector

0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

2) Generate two individuals from the vector

1	1	1	0	0	1	0	1

1	0	0	0	1	1	0	0

3) Let them compete

Winner

1	1	1	0	0	1	0	1

Loser

1	0	0	0	1	1	0	0

4) Update the probability vector towards the better one

(The bits that are different are in gray color.)

0.5	0.6	0.6	0.5	0.4	0.5	0.5	0.6

5) Repeat step 2) – 4) until the vector has converged

Figure 1: The procedure of the cGA

solution. Therefore, each dimension in the probability vector is updated toward the better solution by adding or subtracting the probability with an updating step size $(1/n)$. In a different bit, we add probability when the winner is 1, and subtract the probability when the winner is 0. E.g. updating step size is 0.1, the probability vector becomes as in step 4. The process of the cGA is repeated until the probability vector has converged.

The concept of the cGA is simple and it has been proved that it performs like the sGA with population n, when the updating step size in the cGA is $1/n$ [1]. The cGA reduces the size and power requirements of the system by representing the population as a probability vector rather than a collection of bit strings.

3. Implementation

In order to utilize the GPU that is specialized for compute-intensive and highly parallel-computation, high degree of thread level parallelism is desirable. Hence, threads are parallelized by elements instead of individuals. Each thread generates a portion of two individuals and updates the same portion of probability vector.

To generate new individual from probability vector, a random number generator is needed; thus, the Linear Congruential random number generator is employed. Every random number generator uses different random seed in order to maintain random characteristic. After generating random numbers, each random number is compared to the probability value in the probability vector to produce the corresponding element of individuals. Each probability in the probability vector is responsible to produce the element of individual in the same position.

The cGA uses binary representation for its solutions, but the problem solutions are encoded in decimal. Thus, the conversion from binary individual to decimal candidate solution is required, and done on GPU by parallel reduction algorithm to reduce the memory transfer time between CPU and GPU. However, the fitness evaluation is still done on CPU because the fitness function is excluded from the cGA.

To get the best performance from GPU, the GPU must be kept busy processing. Hence, reducing memory access latency can improve performance. Avoiding off-chip memory access by using shared memory is a well-known technique to reduce memory access latency. Due to the fact that shared memory is shared among the threads in the same block, a thread that requires data from another thread should group together. Computing each dimension of candidate solution uses several bit of individual from many threads; thus, these threads are put to the same block.

To avoid the branch divergence, which is an undesirable problem in GPU, the updating direction of each element in the probability vector is calculated from the sign of the subtraction of the same position element of individuals and the subtracted fitness values. The updating equation is shown below, where $\vec{P}_i$ is a probability vector of i^{th} generation. The bit string of j^{th} individual is represented by $\vec{I}_j$, and f_j is its fitness value. The update step size is $1/n$, where n is equal to sGA population size.

$$\vec{P}_{i+1} = \vec{P}_i + \frac{1}{n} \cdot \frac{f_1 - f_2}{|f_1 - f_2|} \cdot (\vec{I}_1 - \vec{I}_2)$$

4. Experiments and Results

To evaluate performance of the cGA-GPU, the F_1-F_6 benchmark functions from the CEC'08 competition session on LSGO [5] are selected because an error of solutions is measurable. The details of benchmark functions are shown in the table below. To minimize the performance variation, these fitness functions are done on CPU because the different implementation of fitness functions also affects the overall performance.

Table 1: Details of benchmark functions from the CEC'08 competition session on LSGO [5]

	Name	Type	Separability
F_1	Shifted Sphere Function	Unimodal	Separable
F_2	Shifted Schwefel's Problem 2.21	Unimodal	Non-separable
F_3	Shifted Rosenbrock's Function	Multi-modal	Non-separable
F_4	Shifted Rastrigin's Function	Multi-modal	Separable
F_5	Shifted Griewank's Function	Multi-modal	Non-separable
F_6	Shifted Ackley's Function	Multi-modal	Separable

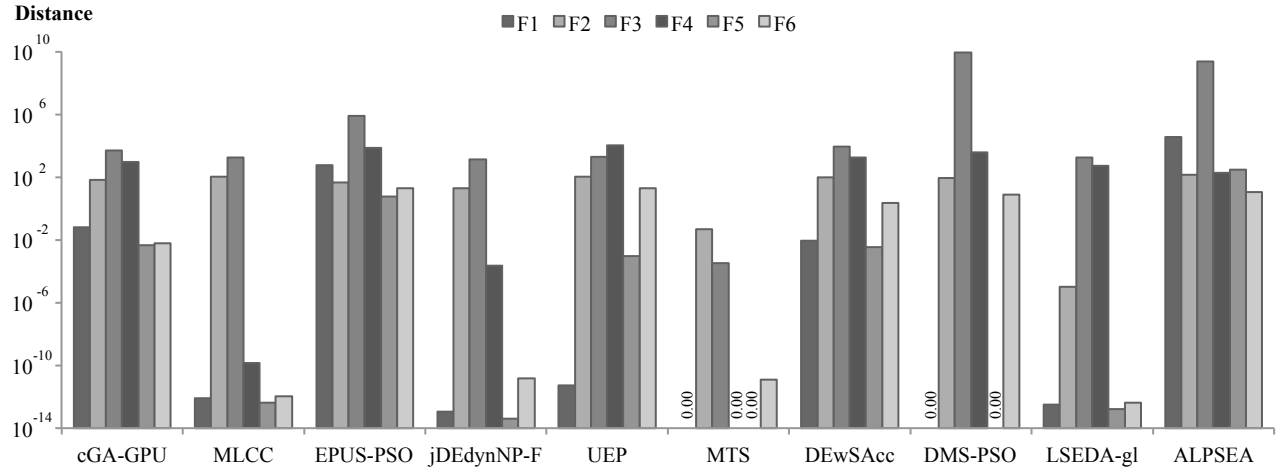

Figure 2: Average distance from optimal fitness value of solutions found by cGA-GPU and other algorithms [5] on 1,000-dimension problems after 5 million function evaluations

Each dimension of solution vector is represented by 256-bit fix-point binary to provide enough precision, and the update rate of each probability is 1/1,000. Every experiment was done on Intel Core 2 duo T8300 and NVIDIA GeForce 8600M GT.

To optimize the GPU efficiently, the number of threads execute concurrently is constrained by the GPU architecture. In this work, the kernel is organized by 64 blocks of threads. Each block contains 128 threads.

Figure 3 shows that the cGA-GPU runs faster than the cGA-CPU when the problem size is increasing. The cGA-GPU achieves an average speedup of 34.45 over the cGA-CPU on 50,000-dimension problems, and it tends to increase in higher dimension problem.

While achieving a great performance speedup, the quality of the solutions found by the cGA-GPU is also acceptable. Comparing to other algorithms in the CEC'08 competition, the quality of the solution found by the cGA-GPU is in the middle as shown in figure 2.

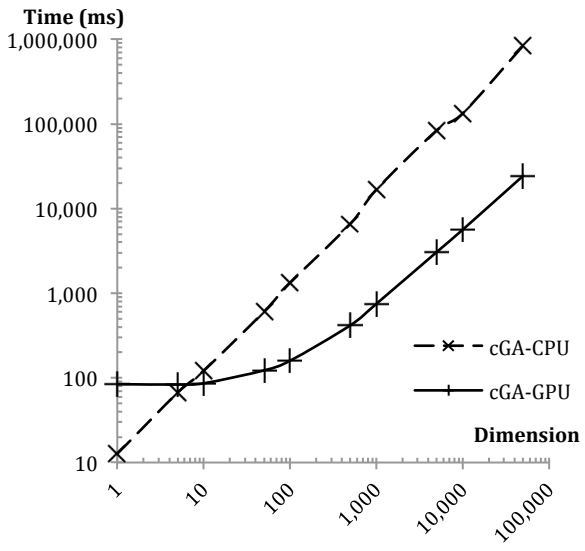

Figure 3: Average execution time of cGA-CPU and cGA-GPU on different problem sizes for 1,000 generations

5. Conclusion

In this paper, we apply the GPU to improve the computation time of the cGA on LSGO problems. The cGA is suitable for GPU because of the independency of each probability in the probability vector. Therefore, our implementation achieved an average speedup of 34.45 on 50,000-dimension problems. In addition, the speedup is increasing when the problem size is increased. However, the lower dimensional problems may not be effectively executed on GPU because of the lower degree of parallelism. Moreover, the cGA-GPU can quickly find acceptable solutions; thus, combining a good quality approach with a fast approach, such as cGA-GPU, should improve both solution quality and performance for solving the LSGO problems.

References

[1] G. R. Harik, F. G. Lobo, and D. E. Goldberg, "The compact genetic algorithm," *IEEE Transactions on Evolutionary Computation*, vol.3, issue 4, pp.287-297, 1999.

[2] C. Aporntewan, and P. Chongstitvatana "A hardware implementation of the compact genetic algorithm," *In Proceedings of the 2001 IEEE Congress on Evolutionary Computation*, 2001.

[3] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A Survey of General-Purpose Computation on Graphics Hardware," *Eurographics 2005*, pp. 21-51, August 2005.

[4] Q. Yu, C. Chen, and Z. Pan, "Parallel Genetic Algorithms on Programmable Graphics Hardware," *Lecture Notes in Computer Science*, vol.3612, 2005.

[5] K. Tang, X. Yao, 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," Technical Report, Nature Inspired Computation and Applications Laboratory, USTC, China, http://nical.ustc.edu.cn/cec08ss.php, 2007.