Advanced Science and Technology Letters
Vol.65 (Database 2014), pp.36-40
http://dx.doi.org/10.14257/astl.2014.65.09

Performance Comparison of GPUs with a Genetic Algorithm based on CUDA

Jae-Hyun Seo[1], Eun-Sol Ko[2], Yong-Hyuk Kim[1]

[1] Department of Computer Science
Kwangwoon University, Seoul, Republic of Korea
[1] {delphia, yhdfly}@kw.ac.kr, [2] kkonsol@nate.com
corresponding author: Yong-Hyuk Kim

Abstract. Generally genetic algorithm (GA) has disadvantage of taking a lot of computation time, and it is worth reducing the execution time while keeping good quality and result. Comparative experiments are conducted with one CPU and four GPUs using CUDA (Compute Unified Device Architecture) and generational GA. We implement the fitness functions of the GA which are suitable for the each environment of the CPU and the GPUs for performance comparison. When experimenting with the CPU, we handle the individual one by one. On the other hand, when experimenting with the GPU, we handle all individuals concurrently. And then we compare and analyze each result of our GA and each time required to process fitness function. There was not a huge difference between the results of the CPU experiment and the GPU ones. In the case of the analysis of computation time, the memory bandwidth of the GPU affects the computation time of fitness evaluation. The numbers of genes that can be processed at the same time are increased by the growth of the clock rate and the number of cores of GPU.

Keywords: CUDA, genetic algorithm, function optimization, parallel GA.

1 Introduction

The complicated algorithms such as evolutionary computation and big data analysis require a high performance, and the processors with multi-cores are in great demand. The usage of OpenCL and CUDA is gradually growing because GPU shows a high efficiency based on thousands of computing cores. In particular, the popularity of CUDA is increasing because it has several advantages such as simple implementation, rapid advances in technology, and good performance. In this study, we use CUDA in fitness evaluation of a genetic algorithm (GA) which is one of the evolutionary computations. Also we compare the performances of CPU and various nVidia Geforce GPUs. We expect that the result of this study can be helpful to other studies that experiment with CUDA and GA.

There have been a number of other studies for a parallel GA using CUDA ass follows. Pospichaland et al. [1, 2] designed the Parallel island genetic algorithm mapping CUDA software model. Various benchmark functions are used as the fitness function, and the result showed that it had 7,000 times improved computation time than the experiments with the nVidia Geforce 8800GTX and the nVidia Geforce

ISSN: 2287-1233 ASTL

GTX285. nVidia Geforce GTX285 had 7,760 times improved computation time than the CPU with the Michalewicz benchmark function. The test results of the nVidia Geforce 8800GTX showed 3,957 times improved performance.

Munawar et al. [1, 3] solved the problem of the MAX-SAT by parallelizing a hybrid genetic algorithm and local optimization based on nVidia Tesla C1060. They explored different genetic operators that could be used for an efficient implementation of parallel GAs. And they obtained a speedup of 25 times in the results of various fitness evaluations. They used a hybrid of cellular GA (cGA) and hill-climbing.

The remainder of this paper is organized as follows: Section 2 explains the test functions used in our experiments. Section 3 describes the environment of our experiments and Section 4 analyzes the results. The paper ends with conclusions in Section 5.

2 Problem Definition

This is the way we find the minimum value with various functions to evaluate performance: Table 1 [4] shows the functions used in our test. These functions are well known, so they are used in various experiments with genetic algorithms [4, 5].

Table 1. Test Functions [4]

Function	*n*	Range of x_i:[l_i, U_j]	Optimum
$F_3 = \sum_{i=1}^{n} \lfloor x_i \rfloor$	256	[-5.12, 5.11]	$x_i \in [-5.12, -5.0)$ for each i
$F_{Schwefel} = \sum_{i=1}^{n} -x_i \sin\sqrt{\lvert x_i \rvert}$	256	[-512, 511]	(420.968746,…, 420.968746)
$F_{M-Sphere} = \sum_{i=1}^{n} x^2$	256	[0, 5.11]	(0, 0,…, 0)
$F_{Sphere} = \sum_{i=1}^{n} x^2$	256	[-5.12, 5.11]	(0, 0,…, 0)
$F_{M-Rastrigin} = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10)$	256	[0, 5.11]	(0, 0,…, 0)
$F_{Rastrigin} = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10)$	256	[-5.12, 5.11]	(0, 0,…, 0)

3 Test Environment

We use the Intel Core2 duo quad core 3.0 GHz, 4GB memory, and the nVidia Geforce GPUs of Table 2 to do various experiments.

Table 2. Specifications of GPU

	GTX260	GT430	GTX760	GTX770
GPU Clock (MHz)	576	700	980~1033	1046~1085
CUDA cores	192	96	1152	1536
Memory Clock	999MHz	800~900MHz	6.0Gbps	7.0Gbps
Memory BandWidth (GB/sec)	111.9	25.6~28.8	192.2	224.3
Memory Interface Width (bit)	448	128	256	256

Table 3 has the parameters of genetic algorithm used in these experiments. The experiments were repeated 30 times. When calculating the fitness function on the GPU, all individuals of population are evaluated concurrently. Also, the operation

time to compute $\sum$ is diminished by using reduction [6]. We set one gene per one block, and each x variable in the gene is allocated in each thread. Each function is calculated in each thread, and the results of the calculation are added up by performing reduction.

Table 3. Parameters in GA

Generational GA parameters			
#Population	500	#Generation	100
Encoding	Real number (256 dimensions)		
Selection	Tournament		
Mutation	Gene-wise (P=0.05), α=0.05		
Crossover	Extended box crossover [8]		
Replacement	If an offspring is superior to the worst individual in the population, we replace it with the worst one.		
Elitism	Use		
Fitness function	Test functions of Table 1.		

4 Test Results

The test results are the average of every result after running 30 times. Fig. 1 shows the comparison of the computation time for each function. In Random experiment, 50,000 random solutions are generated and the solutions are calculated by each test function based on the CPU. The GA parameters of Table 3 are applied to all experiments. The CPU and the GPU test showed better results than Random, and each test result of the CPU and the GPU was similar.

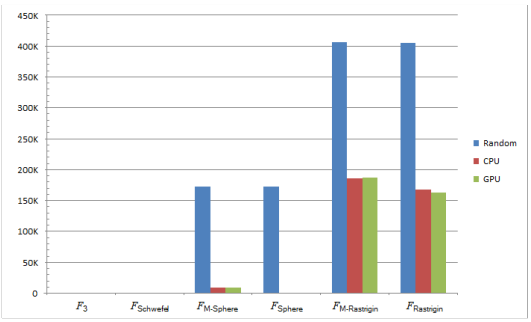
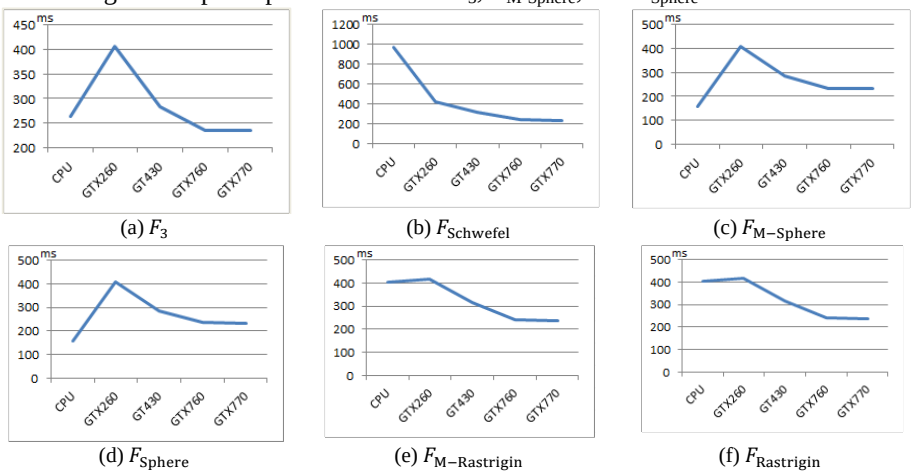


Fig. 1. Comparison of computation time

Fig. 2 shows the computation time of each function. In the analysis of the computation time, the memory bandwidth of the GPU had a strong influence on the computation time when data is copied from the PC memory to the GPU memory. Also, as the clock rates and the number of cores of the GPU increase, the number of genes that can be processed concurrently increases.

There are advantages in using the GPU only when the fitness function is complicated such as $F_{Schwefel}$, $F_{M\text{-}Rastrigin}$, and $F_{Rastrigin}$. However, the computation times of F_3, $F_{M\text{-}Sphere}$, and F_{Sphere} are shorter in CPU experiments. The reason is as follows. In GPU experiments, there is the process of copying data from the PC memory to the

GPU memory. The time spent copying memories is longer than the computation time when using the simple experiment such as F_3, $F_{\text{M-Sphere}}$, and F_{Sphere}.



(a) F_3 (b) F_{Schwefel} (c) $F_{\text{M-Sphere}}$

(d) F_{Sphere} (e) $F_{\text{M-Rastrigin}}$ (f) $F_{\text{Rastrigin}}$

Fig. 2. Comparison of computation time taken for fitness evaluation

5 Conclusions

We conducted comparative experiments with CPU and GPUs under CUDA and GA environments. This study provides the guide in helping someone who wants to use CUDA and GA. In the analysis of the results, there was not a significant difference between the CPU experiment and the GPU ones. However, the memory bandwidth of the GPU has a considerable influence on the computation time. GPU clock rate and the number of GPU core were directly proportional to the number of individuals that could be processed concurrently. The computation time was shorter as the result of these reasons. Future work includes the implementation of all genetic operators using CUDA, and the performance will be promising.

References

1. J.-H. Seo and Y.-H. Kim, "A Survey on Parallel Evolutionary Computation using CUDA," In Proceedings of KIISE Conference, pp. 722–724, 2014. (in Korean)
2. P. Pospichal and J. Jaros, "GPU-based acceleration of the genetic algorithm," GECCO competition, 2009.
3. A. Munawar, M. Wahib, M. Munetomo, and K. Akama, "Hybrid of genetic algorithm and local search to solve max-sat problem using nVIDIA CUDA framework," Genetic Programming and Evolvable Machines, vol. 10, no. 4, pp. 391–415, 2009.
4. Y. Yoon and Y.-H. Kim, "Geometricity of genetic operators for real-coded representation," Applied Mathematics and Computation, vol. 219, no. 23, pp. 10915–10927, 2013.
5. Y. Yoon, Y.-H. Kim, A. Moraglio, and B.-R. Moon "A Theoretical and Empirical Investigation on the Lagrangian Capacities of the 0/1 Multidimensional Knapsack Problem,

European Journal of Operational Research," Information Sciences, vol. 183, no. 1, pp. 48–65, 2012.
6. M. Harris, "Optimizing parallel reduction in CUDA," NVIDIA Developer Technology, vol. 2, no. 4, 2007.

