Differential Evolution on a GPGPU: The Influence of Parameters on Speedup and the Quality of Solutions

Omar A. C. Cortes*, Mônica S. Pais[†], Filipo N. Mór[‡], Andrew Rau-Chaplin[§] and César A. M. Marcon[‡]
*Instituto Federal do Maranhão
São Luis, MA, Brazil
Email: omar@ifma.edu.br
[†]Informatics Department
Instituto Federal Goiano
Urataí, GO, Brazil
Email: monica.pais@ifgoiano.edu.br
[‡]Faculty of Computer Science
Dalhousie University
Halifax, NS, Canada
Email: arc@cs.dal.ca
[§] Department of Computer Science
Pontifícia Universidade Católica
Porto Alegre, RS, Brazil
Email: filipo.mor@acad.pucrs.br, cesar.marcon@pucrs.br

***Abstract*—One challenge in studying the speedup performance of evolutionary optimization techniques, particularly in differential evolution, is that many parameters including crossover rate, F, dimensionality, population size and the complexity of the objective function play an important role. In fact, these same parameters also effect the quality of the obtained results. Therefore, it is important to understand the interaction between these parameters in order to make good choices for these key parameters that drive both the quality and speedup metrics. Thus, the purpose of this paper is to show how parameters such as crossover rate, F, dimension, population size, and calls to evaluation functions can influence the speedup and the quality of solutions in a differential evolution algorithm in high dimension problems. The evaluation was done using a 2^k factorial analysis considering a Schwefel Benchmark Function in a Matlab implementation running on a general purpose GPU. Results have shown that a reasonable speedup can be reached taking into account a high level of programming, *i.e.*, there are a good trade-off between the required effort to program on GPU in Matlab and the reached Speedup. On the other hand, results in terms of quality of solutions showed that CPU tends to produce better outcomes in some configurations.**

***Keywords*-Differential Evolution; Speedup; GPU; Factorial Analysis;**

I. Introduction

Differential evolution [1] (DE) is an evolutionary optimization technique that has proven to be highly effective in a wide range of applications. It is particularly effective when dealing with optimization in complex applications such as reinsurance treaty optimization [2], wave form inversion [3], vibration suppression controller [4], scheduling of a flexible assembly line [5], etc. Given its widespread use in applications and often considerable run times it is interesting to study its performance on parallel hardware, especially in alternative technologies such as graphic processing units (GPU) because nowadays even personal computer can come equipped with GPGPU (General Purpose GPU) making this kind of architecture easily available. In fact, the processing power by watt offered by this kind of architecture is very attractive [6].

In this context, it is important to notice that the DE algorithm has its data structure mostly based on arrays, therefore, it is suitable for execution on GPGPUs. So, in order to use the capabilities of a GPGPU some languages are available to different levels of programmers. The first one is C-CUDA (Compute Unified Device Architecture) [7] where programmers can extract all capabilities of parallelization offered by this kind of architecture, maximizing the obtained speedup. On the other hand, this kind of programming language demands an expressive knowledge about the architecture, its capabilities and how to use it. A second approach and more suitable to application programmers is the use of Matlab which allows us to program parallel applications on GPU at a very high level using the parallel computation toolbox [8]; however, this approach might lead to a decrease in the speedup that can be reached.

Thus, we investigate how changing parameters in a DE algorithm can affect the speedup considering a high level approach of programming. In order to do so, we analyzed the effect of the following parameters: crossover rate, population size, dimension, number of calls to the evaluation function, and multiplication factor. The first one is regarded to elements being moved from one matrix to another. The second and third ones are related to the size of data structure required to store the population and solutions, and apply operations on, so affecting the speedup. The forth one regards to the number of times that calls to the evaluation function are done, particularly in GPU these calls are executed inside the GPU, therefore affecting the speedup as well. Finally, the fifth one can affect the quality of solutions. The quality of solutions are analyzed considering five parameters: architecture, crossover rate, multiplier factor, population size and calls to the evaluations function.

Both analysis are done by means of design of experiments (DOE) [9] which is a powerful technique used for both exploring new processes and gaining increased knowledge of existing ones, followed by optimizing theses processes for achieving good performance [10]. Actually, all evaluations were done using a 2^k factorial design with four and five factors, and two levels, *i.e.*, 2^4 treatments which allow us to identify how each factor and their interaction influence on gaining speedup.

The remainder of this paper is divided as follows: Section II describes the related works on DE and briefly presents the main problems in evaluating results; Section III illustrates the DE algorithm and how it works; Section IV shows the necessary changes into the code, the benchmark being used in the experiments, and the factorial analysis; finally, Section V illustrates the conclusions of this paper and also presents the future work.

II. Related Works

The attempt of improving the speedup on DE applications using parallel computing is not new. Tesoulis et. al. [11] presented one of first attempts to improve the speedup on DE using a ring topology in a parallel distributed system. Since then, many investigations have appeared, including those one involving GPUs.

The first versions of DE on GPGPUs were introduced in 2010 by de Veronese and Krohling [12], Zhu [13], and Zhu and Li [14]. All of those works were implemented in C-CUDA and used benchmarks functions in order to show how much speedup can be reached. Moreover, particular DE applications on GPGPU have been investigated in works such as [15], [16], [17], and [18]. In this context, reports have revealed speedups between 1.5 and 120 according to Krömer et. al. [19].

Regardless the problem being solved using DE, many researches have no planning on their experiments, consequently making impossible to reproduce them [20]. These issues have been addressed by authors such as Rooker [22] and Rardin [23]. In other words, it is very common to find out researches which investigate the performance of a particular evolutionary algorithm without using a proper statistical analysis. On the other hand, works such as [24] and [25] used a proper statistical background in their analysis making those experiments reproducible.

Thus, as we can notice, statistical tools for analyzing the performance of evolutionary algorithms have not been broadly used neither systematized into methodologies and/or frameworks. So, in order to fill up this gap, Pais started proposing a first framework to evaluate parallel genetic algorithms in [27]. This work ended up in a doctoral thesis [21] in which a complete methodology for experimentation in parallel evolutionary algorithms based on multi-core architectures was published, consequently producing works such as [20] and [26]. Then, Cortes [28] applied part of this methodology in order to discover what parameters influence on speedup of a Particle Swarm Optimization (PSO) [29] algorithm running in a GPGPU.

At the best of our knowledge, there are no other works applying Pais' methodology neither in evolutionary algorithms nor in parallel evolutionary algorithms, particularly there are no applications of her methodology in GPU systems other than that one presented in [28] which is a different algorithm. Hence, this work applies Pais' methodology in order to discover which parameters have more influence on achieving speedup in a differential evolution algorithm executed in a GPGPU with less programming effort using Matlab. As a consequence of using Pais' methodology our experiments are reproducible.

III. Differential Evolution

The DE algorithm was proposed by Storn [1] in 1995, being based on the difference between two individuals which is summed up to a third one. The process is sketched in the Algorithm 1, where the first step is to create a population at random. Then, while the stop criteria is not reached the vector of differences is calculated according to the Equation 1 , where idx_i is a vector with three individual randomly chosen, and F is a multiplication factor normally between 0 and 1. This strategy is called $DE/Rand/1/bin$ because Pop_{idx_3} is randomly chosen. When P_{idx_3} is the best individual in the population the strategy is called $DE/Best/1/bin$.

$$v = Pop_{idx_3} + F * (Pop^k_{idx_1} - Pop^k_{idx_2}) \tag{1}$$

The process of computing v is called mutation. Afterwards, a new individual is created in a similar way than the discrete crossover of genetic algorithms, *i.e.*, for each dimension d a gene is chosen from the vector of differences v with a probability of CR, or from the target individual i with a probability of $1 - CR$. Finally, if the fitness of the new individual is better than the fitness of the target one then the new individual, called $indiv$, replaces the individual in the population pop_i.

```
$Pop \leftarrow$ generate_pop(n,d)
$fit \leftarrow$ evaluate_($P^k$)
while (Stop Criteria is FALSE) do
    for i = 1 to #pop_size do
        $idx \leftarrow$ select_indiv(3)
        $v \leftarrow Pop_{idx_3} + F * (Pop^k_{idx_1} - Pop^k_{idx_2})$
        for j = 1 to dimension do
            nj = rand()
            if (nj < CR) then
                $indiv_j \leftarrow v_j$
            else
                $indiv_j \leftarrow pop_{ij}$
            end
        end
        $fit' \leftarrow$ evaluate($indiv$)
        if $fit' < fit_i$ then
            $pop_i \leftarrow indiv$
            $fit_i \leftarrow fit'$
        end
    end
end
```

Algorithm 1: Canonical Differential Evolution Algorithm

```
$pop \leftarrow$ generate_pop_on GPU(pop_size,dimension);
$fit \leftarrow$ evaluate_on_GPU(pop);
while (#Calls to Evaluation Function is not Reached) do
    $r1 \leftarrow$ matrix_rand_indexes(pop_size,3);
    $v \leftarrow$ pop[r1[:,3]] + F .* (pop[r1[:,2]] - pop[r1[:,1]]);
    $idx \leftarrow$ (v > upper_bounds);
    $v[idx] \leftarrow$ upper_bounds;
    $idx \leftarrow$ (v < lower_bounds);
    $v[idx] \leftarrow$ lower_bounds;
    $r2 \leftarrow$ matrix_rand_crossover(pop_size,dimension);
    $idx \leftarrow$ (r2 < CR);
    $pop'[idx] \leftarrow$ v[idx];
    $pop'[!idx] \leftarrow$ pop[!idx];
    $fit' \leftarrow$ evaluate_on_GPU(new_pop);
    $idx \leftarrow$ (fit' < fit);
    $pop[idx,:] \leftarrow$ pop'[idx,:];
    $fit[idx] \leftarrow$ fit'[idx];
end
```

Algorithm 2: Differential Evolution Algorithm on GPU

IV. Results

A. Changes in the Algorithm

Programming in a high level language usually rise up two conflict features in parallel computing using GPUs. On one hand, it can reduce the effort in programming complex applications. On the other hand, it is not possible to control the GPU execution without recurring to C-CUDA programming. Thus, it is important to know how to extract the better characteristics of the GPU programming in a high level of coding.

In this context, the following practices are important in order to have a better fit with the GPGPU hardware: (i) using array as much as possible in GPU memory; (II) executing functions on GPU; and, (iii) using logical indexes in operations with arrays. So, taking that practices into account the DE algorithm is transformed into that one presented in Algorithm 2. A side benefit of this approach is that it allows applications to be coded in a high-level language like Matlab. An important detail to be aware about is that on GPU indexation such as $pop[i, j]$, for example, is completely forbidden.

So, taking into account that the algorithm was implemented in Matlab, the first step of DE on GPU is to create the population at random on GPU RAM which can be done using the function $gpuArry.rand()$. The evaluation process has to be executed on GPU as well, using preferably the function $arrayfun(fun, A1, ..., An)$, where fun represents the evaluation function to be called and A the respective parameters.

Afterwards, a matrix $r1$ is computed storing all indexes that will be used to create the mutation vector v, which is created at once for the entire population, *i.e.*, the computation of the variable v is now a matrix operation, where the symbol $.*$ means an element-wise multiplication. Then, the boundaries for each dimension are verified using logical indexing; if one or more elements violate this constraint they are saturated on the respective limit of the domain.

Thereafter, a random matrix used in the crossover process is randomly created, where idx contains all indexes in which $r2 < CR$, *i.e.*, those indexes whose genes will come from v. Whereas, $!idx$ consists of all indexes that will come from the current population pop. Finally, the new population is evaluated on GPU and the individuals who present a better fitness than current population will replace them going to a next iteration.

B. Experiments

Two set of experiments have been conducted in order to analyze the impact of using GPU on speedup and quality of solutions of DE algorithm. Each set was analyzed using a factorial analysis [9] with four and five factors, respectively. Also, each treatment was executed 50 times in order to try to guarantee the data normality, summing up 800 runs for the first set of experiments related to speedup and 1600 runs for the second one regarded to quality of solutions. The experiment has been implemented and executed in Matlab version 2013 using the Parallel Computing Toolbox on a Windows 7 64-bit Operating System running on an Intel i7 3.4 Ghz processor, with 16 GB of RAM and a NVIDIA GPU GTS450. The factorial analysis was done using Mintab version 17.

The problem being solved is a multimodal (it has many local optima being hard to solve) benchmark function called Schwefel, which is computed according to Equation 2. The referred function has to be minimized, so assuming that x is a vector of parameters or variables and f is the function that we want to minimize, the problem can be mathematically written as $min_{x \in \mathbb{R}} \ f(x)$, where x is in a continuous space.

In this specific problem the domain for each dimension is within [-500, 500] and the minimum is calculated by $f(x) = -n \times 418.9829$, where n is the problem dimension. In this context, the minimums for each level according to the parameter dimension presented in Table I are -41898.29 and -837965.8, respectively. Figure 1 shows a bi-dimensional search space of Schwefel functions, where can see many local optima.

$$f(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{x_i}) \tag{2}$$

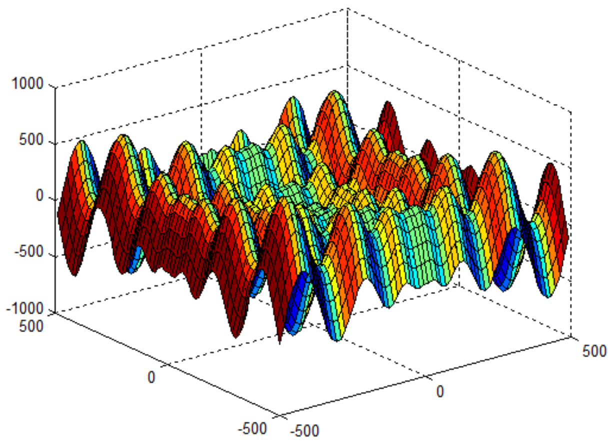

Figure 1. Bi-dimensional Schwefel function

1) Speedup: In order to evaluate the speedup, a factorial analysis with four factor and two levels has been considered in this experiment, where the following parameters were selected: crossover rate (CR), dimension (*Dim*), number of calls to the evaluation function (*Calls*) and population size (*PopSize*). These parameters were chosen because they might influence in obtaining the speedup since data structures are larger, which can be a benefit in terms of speed in a GPU. The experiment factors and their respective levels are shown in Table I. Moreover, the parameter F was set to 0.7 in both DE algorithms (CPU and GPU) while speedup was evaluated.

Table I
EXPERIMENT FACTORS AND LEVELS

Levels	CR	Dimension	Calls	PopSize
Low Level (-1)	0.3	100	2×10^5	500
High Level (1)	0.9	2000	10^6	3000

The experiment design, the speedup and the standard deviation are shown in Table II, where as previously mentioned the speedup represents the average of 50 executions on each treatment. The optimum configuration is $CR = 0.9$, $Dim = 2000$, $Calls = 10^5$, and $PopSize = 500$ which corresponds to treatment 13 leading to a speedup of 4.9351. Three sub-optimal configuration were found in treatments 8, 9 and 11. Even though these configuration are similar, a t-test between them using an $\alpha = 0.05$ indicates that these differences are meaningful between treatments 13 and 8, and they are not meaningful between treatments 9 and 11; therefore, the first configuration is really the best one.

The full model obtained by the factorial analysis is presented in Equation 3. Then a second-order linear regression model obtained by an automatic search procedure based on the AIC criterion results in the adjusted model is shown in Equation 4, where non-significant effects were removed from the model. This last adjusted model presents a residual standard error of 0.588 on 4 degrees of freedom, and R^2 was 0.928, which means that the model explains 92.8% of the variation of the speedup. The model adequacy and its assumptions were checked. The residuals versus predicted values plot did not present a pattern or a structure and the normal probability plot resembled a straight line. Also formal tests of independence, equality of variances and normality of residuals were performed.

$$\begin{aligned} Sp = \; & 3.1162 + 0.4654\mathrm{CR} - 0.1214 Dim - \\ & 0.1615 Calls - 0.1319 PopSize - \\ & 0.3086\mathrm{CR}*Dim - 0.3361\mathrm{CR}*Calls - \\ & 0.3664\mathrm{CR}*PopSize - 0.1102 Dim* \\ & Calls + 0.2735 Dim*PopSize + \\ & 0.4091 Calls*PopSize - 0.5011\mathrm{CR}* \\ & Dim*Calls - 0.09165\mathrm{CR}*Dim* \\ & PopSize - 0.1767\mathrm{CR}*Calls*PopSize + \\ & 0.2027 Dim*Calls*PopSize - \\ & 0.07676\mathrm{CR}*Dim*Calls*PopSize \end{aligned} \tag{3}$$

$$\begin{aligned} Sp = \; & 3.1162 + 0.4654\mathrm{CR} - 0.1214 Dim \\ & -0.1615 Calls - 0.1319 PopSize - \\ & 0.3086\mathrm{CR}*Dim - 0.3361\mathrm{CR}*Calls - \\ & 0.3664\mathrm{CR}*PopSize - 0.1102 Dim* \\ & Calls + 0.4091 Calls*PopSize - \\ & 0.5011\mathrm{CR}*Dim*Calls \end{aligned} \tag{4}$$

In order to make more inferences about how those parameters influence on speedup we have to look into some factorial analysis, especially into adjusted model since it indicates which parameters are significant in obtaining the speedup. Further, charts that were created by the analysis are shown in Figure 2 which represents a pareto plot, the main effect of factors and the interaction between factors. The pareto chart indicates that interactions between two factors might be more important in obtaining the speedup; Even though all of them are located before the reference line, the adjusted model (Equation 4) presents all significant main and interaction effects. Then, we can identify if the influence of factors is positive or negative in the main effect figure which

Table II
DESIGN, MEAN SPEEDUP AND STANDARD DEVIATION

Treat.	CR	Dim	Calls	PopSize	Speedup	St. Dev.
1	-1	-1	-1	-1	3.1170	0.0032
2	-1	-1	-1	1	2.2430	0.0057
3	-1	-1	1	-1	2.0716	0.0582
4	-1	-1	1	1	2.4230	0.0269
5	-1	1	-1	-1	2.5380	0.0202
6	-1	1	-1	1	2.0070	0.0268
7	-1	1	1	-1	1.9386	0.0269
8	-1	1	1	1	**4.8683**	0.0553
9	1	-1	-1	-1	**4.6846**	0.0365
10	1	-1	-1	1	3.1114	0.0061
11	1	-1	1	-1	**4.6990**	0.0311
12	1	-1	1	1	3.5517	0.0036
13	1	1	-1	-1	**4.9351**	0.0754
14	1	1	-1	1	3.5857	0.0174
15	1	1	1	-1	2.0008	0.0119
16	1	1	1	1	2.0845	0.0149

indicates that calls to evaluation function has a positive effect on speedup, whereas the other ones have a negative impact because they tend to reduce the speedup when we change the respective parameters from low level to high level.

The adjusted model shows significant interaction effects, therefore, it is necessary to identify how the speedup changes according to it. With CR set to 0.3, when Dim changed from 100 to 2000, the mean speedup increase by 0.1872. With CR set to 0.9, the mean speedup decreases by 0.4301 as the Dim changed from 100 to 2000. When CR is set to 0.3, when $Calls$ changed from 2×10^5 to 10^6, the mean speedup increases by 0.1745. With CR set to 0.9, the mean speedup decrease by 0.4976 as the $Calls$ changes from 2×10^5 to 10^6. The effect of the interaction $Calls * PopSize$ was 0.4091. With $Calls$ set to 2×10^5, when $PopSize$ changed from 500 to 3000, the mean speedup decreased by 0.5409. With $Calls$ set to 10^6, the mean speedup increased by 0.2772 as $PopSize$ changed from 500 to 3000. With CR set to 0.3, when $PopSize$ changed from 500 to 3000, the mean speedup increased by 0.2345, whereas with CR set to 0.9, the mean speedup decrease by 0.4983 as the $PopSize$ changes from 500 to 3000.

In terms of dimension, with Dim set to 100, when $Calls$ changed from 2×10^5 to 10^6, the mean speedup decreased by 0.05135. Then, with Dim set to 2000, the mean speedup decrease by 0.2717 as the $Calls$ changes from 2×10^5 to 10^6. With Dim set to 100, when $PopSize$ changed from 500 to 3000, the mean speedup decreases by 0.4054. With Dim set to 2000, the mean speedup increases by 0.1416 as the $Calls$ changes from 500 to 3000. Finally, the three factors interaction $PopSize * Calls$ indicates with CR set to 0.3, the interaction effect of $Dim * Calls$ is 0.3909, and with CR set to 0.9, the interaction effect of $Dim * Calls$ is -0.6113 on speedup.

All in all, keeping CR in low level and then $PopSize$, $Calls$ and Dim in high levels tend to present higher speedups, which makes sense because the algorithm fits better in the GPGPU architecture. On the other hand, other combinations can also reach goods speedups.

2) Quality of Solutions: In order to evaluate the quality of solutions some changes are mandatory in the previous factorial design. The first one is related to dimension which has to be fixed because the optimal solutions has to be the same for every configuration, therefore the factor Dim was set to 1000. The second one is to add the factor F (with levels 0.3 and 0.7) in the analysis which can also affect the quality of a solutions because it changes the ability of exploration and exploitation of the algorithm. Finally, the last modification that has to be done is to add the architecture ($Arch = \{CPU, GPU\}$) as a factor, thus we can identify which one produces solutions with better quality.

Table III presents the design, the mean of the 50 executions and the standard deviation, where we can observe that the best solution was found by the following configuration: $Architecture = CPU$, CR=0.3, F=0.7, $Calls = 10^6$, $PopSize = 500$. After these changes it is important to notice that in this particular case a decreasing curve has a positive effect because the global optima is a negative value.

Figure 3 shows the factorial plots created with the results from Table III. So, looking at the pareto chart, we can state that the factor $Calls$ and the interaction between factors (excepting the interactions $CR * Calls$, $F * Calls$ e $F * PopSize$) are potentially the most important in obtaining quality in solutions. On the other hand, the variance presented by the results does not allows us to create a model which reflects its behavior. Thus, a two tailed t-test comparing CPU and GPU, using $\alpha = 0.05$ ($-2.93 < t < 2.93$), and considering that all means are the same upon the same configurations is presented in Table III, where we can observe that solutions in CPU (underlined) tends to be better than in GPU when CR is set to 0.3, whereas solutions tend to present the same quality when CR is set to 0.7. Also,

Table III
DESIGN, MEAN QUALITY AND STANDARD DEVIATION

$Arch$	CR	F	$Calls$	$PopSize$	Quality	Stdev
-1	-1	-1	-1	-1	-99561.22913	8905.992217
-1	-1	-1	-1	1	-39242.5668	1593.551144
-1	-1	-1	1	-1	-102331.3489	8888.703093
-1	-1	-1	1	1	-50781.27507	7912.635914
-1	-1	1	-1	-1	-62364.11626	9274.956581
-1	-1	1	-1	1	-40861.14397	1461.503669
-1	-1	1	1	-1	**-187272.5542**	16969.87567
-1	-1	1	1	1	-51909.33291	2283.717489
-1	1	-1	-1	-1	-33918.06921	12270.31176
-1	1	-1	-1	1	-28024.25462	1843.27535
-1	1	-1	1	-1	-158249.4069	8781.541603
-1	1	-1	1	1	-30806.27824	1381.554256
-1	1	1	-1	-1	-34618.38043	2263.94109
-1	1	1	-1	1	-31080.21761	1544.26524
-1	1	1	1	-1	-59736.3109	13915.89955
-1	1	1	1	1	-35240.80279	1337.493624
1	-1	-1	-1	-1	-46850.86065	1593.477414
1	-1	-1	-1	1	-37403.13661	1612.08002
1	-1	-1	1	-1	-60814.56003	1927.913943
1	-1	-1	1	1	-45950.51283	1378.569687
1	-1	1	-1	-1	-53578.91429	1581.2079
1	-1	1	-1	1	-39611.77496	1425.713312
1	-1	1	1	-1	-70688.96682	1932.46718
1	-1	1	1	1	-52786.99504	1542.721017
1	1	-1	-1	-1	-31024.70727	1629.50563
1	1	-1	-1	1	-28378.46091	1910.322733
1	1	-1	1	-1	-44985.45356	1840.412803
1	1	-1	1	1	-31113.30699	1370.260155
1	1	1	-1	-1	-35291.14047	1673.76496
1	1	1	-1	1	-31162.43624	1558.8105
1	1	1	1	-1	-48298.10131	2049.629688
1	1	1	1	1	-35325.35666	1292.564049

it is important to highlight that it is not possible to create a model in terms of quality since the standard deviation is high.

V. CONCLUSIONS

This paper showed how parameters in a DE algorithm can affect the speedup and the quality of solutions obtained in a CPU and a GPGPU using a high level of programming. Results indicated that the interaction between factors is important when achieving speedup, especially when we take into account the dimension and calls to evaluation function, whereas CPU tends to present better solutions with CR set to low level. Future work includes to analyze the effects of parameters on speedup in a C-CUDA implementation where programmers have more control on threads and blocks being executed in GPU. Also, an analysis using a real world application based on reinsurance analytics is also in mind.

ACKNOWLEDGMENT

The authors would like to thank the Instituto Federal de Educação, Ciência e Tecnologia do Maranhão (IFMA) and Conselho Nacional de Desenvolvimento Científico (CNPq) for funding this research.

REFERENCES

[1] R. Storn, and K. Price, “Differential Evolution: A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces”, Journal of Global Optimization, v. 11, 1997.

[2] O. A. C Cortes, A. Rau-Chaplin, and P. F. do Prado, “ On VEPSO and VEDE for Solving a Treaty Optimization Problem”, IEEE Conference on System, Man, and Cybernetics, San Diego-USA, 2014.

[3] Zhaoqi Gao, Zhibin Pan, and Jinghuai Gao, “ New Highly Efficient Differential Evolution Scheme and Its Application to Waveform Inversion”, IEEE Geoscience and Remote Sensing Letters, v. 11, n.10, pp.1702-1706, 2014.

[4] H. Ikeda and H. Tsuyoshi, “Design of m-IPD controller of multi-inertia system using Differential Evolution”, International Power Electronics Conference, pp. 2476-2482, 2014.

[5] L. W. H. Vincent and S. G. Ponnambalam, “A Differential Evolution-Based Algorithm to Schedule Flexible Assembly Lines”, IEEE Transactions on Automation Science and Engineering, v. 10, n.4, pp.1161–1165, 2013.

[6] S. Potluri, A. Venkatesh, D. Bureddy, K. Kandalla, and D. K. Panda, “Efficient Intra-node Communication on Intel-MIC Clusters”, 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp.128-135, 2013.

Table IV
COMPARIONS CPU VS GPU USING A T-TEST

CR	F	Calls	PopSize	CPU	Stdev	GPU	Stdev	t
-1	-1	-1	-1	-99561.229	8905.99	-46850.860	1593.48	-41.20
-1	-1	-1	1	-39242.567	1593.55	-37403.137	1612.08	-5.74
-1	-1	1	-1	-102331.349	8888.70	-60814.560	1927.91	-32.28
-1	-1	1	1	-50781.275	7912.64	-45950.513	1378.57	-4.25
-1	1	-1	-1	-62364.116	9274.96	-53578.914	1581.20	-6.60
-1	1	-1	1	-40861.144	1461.50	-39611.775	1425.71	-4.32
-1	1	1	-1	-187272.554	16969.88	-70688.967	1932.47	-48.27
-1	1	1	1	**-51909.333**	2283.71	**-52786.995**	1542.72	**2.25**
1	-1	-1	-1	**-33918.069**	12270.31	**-31024.707**	1629.50	**-1.65**
1	-1	-1	1	**-28024.255**	1843.27	**-28378.461**	1910.32	**0.94**
1	-1	1	-1	-158249.407	8781.54	-44985.454	1840.41	-89.26
1	-1	1	1	**-30806.279**	1381.55	**-31113.307**	1370.26	**1.12**
1	1	-1	-1	**-34618.380**	2263.94	**-35291.140**	1673.76	**1.69**
1	1	-1	1	**-31080.217**	1544.26	**-31162.436**	1558.81	**0.26**
1	1	1	-1	-59736.311	13915.90	-48298.101	2049.63	-5.75
1	1	1	1	**-35240.8028**	1337.4936	**-35325.3567**	1292.5640	**0.32**

[7] NVIDIA, NVIDIA CUDA Programming Guide 4.1, Available in http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz3Hpy6xWoR. Visited on September, 2014.

[8] Matlab, "Speeding Up MATLAB Computations with GPUs", Available in http://www.mathworks.com/products/parallel-computing/features.html#speeding-up-matlab-computations-with-gpus, Visited on October, 2014.

[9] D. C. Montgomery, "Design and analysis of experiments", 7th ed., John Wiley and Sons, Hoboken, 2009.

[10] J. Anton, "Design of Experiments for Engineers and Scientists", Waltham-MA: Elsevier, 2014.

[11] D. Tasoulis, N. Pavlidis, V. Plagianakos, and M. Vrahatis, "Parallel differential evolution", Congress on Evolutionary Computation, v. 2, pp. 20232029, 2004.

[12] L. de Veronese and R. Krohling, "Differential evolution algorithm on the GPU with C-CUDA", IEEE Congress on Evolutionary Computation (CEC), pp. 17, 2010

[13] W. Zhu, "Massively parallel differential evolution - pattern search optimization with graphics hardware acceleration: an investigation on bound constrained optimization problems", Journal of Global Optimization, pp. 121, 2010, 10.1007/s10898-010-9590-0. Available: http://dx.doi.org/10.1007/s10898-010-9590-0

[14] W. Zhu and Y. Li, "Gpu-accelerated differential evolutionary markov chain monte carlo method for multi-objective optimization over continuous space", Proceeding of the 2nd workshop on Bio-inspired algorithms for distributed systems, New York, NY, USA, pp. 18, 2010, [Online]. Available: http://doi.acm.org/10.1145/1809018.1809021

[15] M. Simonsen, C. N. Pedersen, M. H. Christensen, and R. Thomsen, "Gpu-accelerated high-accuracy molecular docking using guided differential evolution: real world applications", Conference on Genetic and evolutionary computation, New York, NY, USA: ACM, 2011, pp. 18031810, 2011.

[16] C. Xiao and W. Qiming, "Modied parallel differential evolution algorithm with local spectral feature to solve data registration problems", Computer Science and Network Technology, v. 3, pp. 13861389, 2011.

[17] L. E. Ramirez-Chavez, C. A. Coello Coello, and E. Rodriguez-Tello, "A gpu-based implementation of differential evolution for solving the gene regulatory network model inference problem", Fourth International Workshop on Parallel Architectures and Bioinspired Algorithms, Galveston Island, TX, USA, October, 2011.

[18] A. K. Qin, F. Raimondo, F. Forbes, and Y. S. Ong, "An improved CUDA-based implementation of differential evolution on GPU", in Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference, New York, NY, USA: ACM, pp. 991998, 2012.

[19] Krömer, P. and Platoš, J. and Sńašel, V., "A Brief Survey of Differential Evolution on Graphic Processing Units", IEEE Symposium on Differential Evolution, pp. 157-164, 2013.

[20] M. S. Pais, I. S. Peretta, K. Yamanaka, and E. R. Pinto, "Factorial design analysis applied to the performance of parallel evolutionary algorithms", Journal of the Brazilian Computer Society February, 20:6, Springer, 2014.

[21] M. S. Pais,"Estudo da Influência dos Parámetros de Algoritmos Paralelos da Computação Evolutiva no seu Desempenho em Plataformas Multicore", Phd Thesis, University of Uberlândia, 2014.

[22] J. Hooker, "Testing heuristics: We have it all wrong", Journal of Heuristics, v. 1, pp. 3342, 1995.

[23] R. L. Rardin and R. Uzsoy, "Experimental evaluation of heuristic optimization algorithms: A tutorial", Journal of Heuristics, v. 7, pp.261304, May 2001.

[24] . P. Assis, A. L. Maravilha, A. Vivas, F. Campelo,and J. A. Ramírez, "Multiobjective vehicle routing problem with fixed delivery and optional collections", Optimization letters, v. 7, Issue 7, pp 1419-1431, 2013.

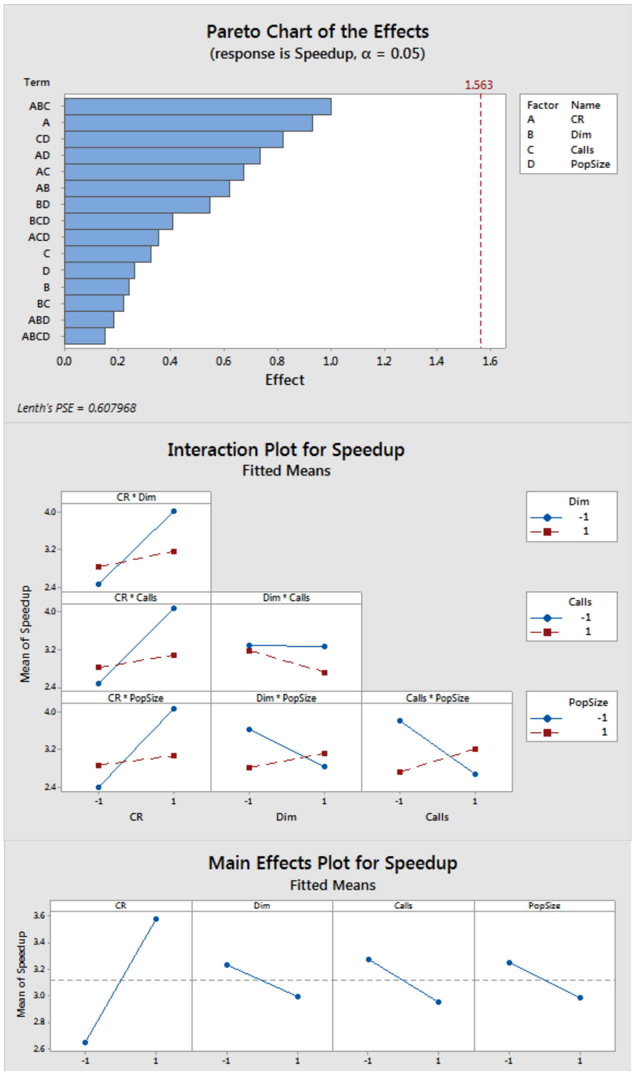

Figure 2. Pareto chart, interaction and main effects plots

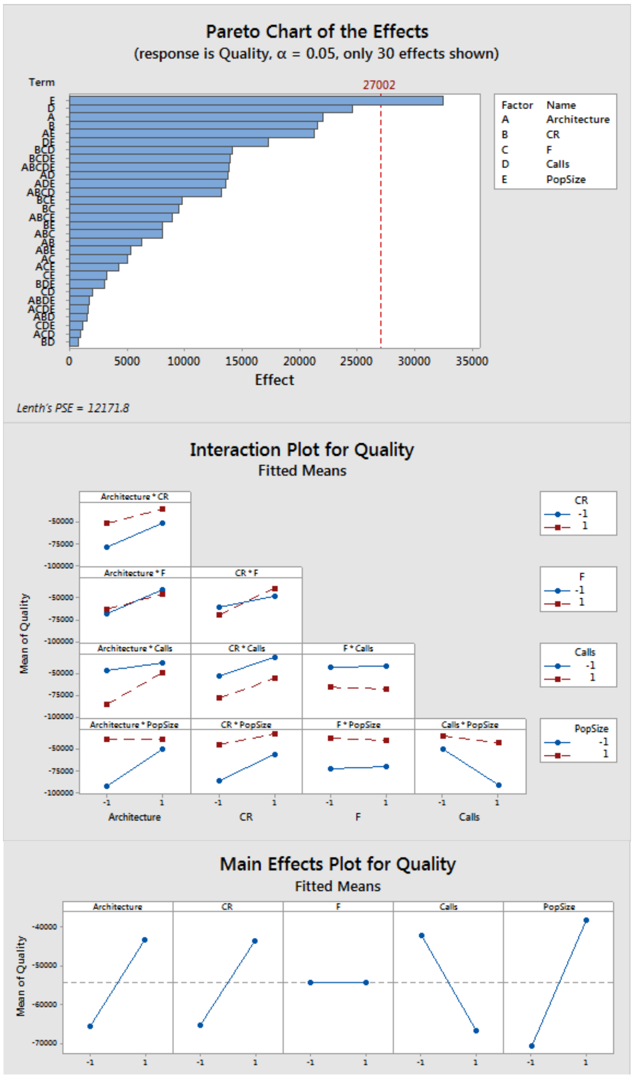

Figure 3. Pareto chart, interaction and main effects plots for quality of results

[25] R. A. Lopes, R. C. Pedrosa Silva, A. R.R. Freitas, F. Campelo, and F. G. Guimares, "A study on the configuration of migratory flows in island model differential evolution", In Proceedings of the conference companion on Genetic and evolutionary computation companion (GECCO), New York, p. 1015-1022, 2013.

[26] M. S. Pais, K. Yamanaka, and E. R. Pinto, "Rigorous Experimental Performance Analysis of Parallel Evolutionary Algorithms on Multicore Platforms", Revista IEEE Amrica Latina, v. 12, Issue 4, 2014.

[27] M. S. Pais, I. S. Peretta, G. F. M. Lima, J. A. Tavares, H. X. Rocha, and K. Yamanaka, "Análise de Fatores Determinantes no Desempenho dos Algoritmos Genéticos Paralelos em Processadores Multinúcleos. In: X Congresso Brasileiro de Inteligência Computacional, 2011.

[28] O. A. C. Cortes and P. F. do Prado, "Avaliação do impacto da dimensão e do número de partículas no speedup em otimização por nuvem de partículas usando GPUs", Escola Regional de Alto Desempenho de So Paulo,2014.

[29] J. Kennedy and J. Eberhart, "Particle Swarm Optimization", IEEE International Conference on Neural Networks, v. 4, 1997.