# Parallelization of the Cuckoo Search Using CUDA Architecture

Raka JOVANOVIC
University of Belgrade
Institute of Physycs
Pregrevica 118, 11000 Belgrade
SERBIA
rakabog@yahoo.com

Milan TUBA
Megatrend University Belgrade
Faculty of Computer Science
Bulevar umetnosti 29, 11070 Belgrade
SERBIA
tuba@ieee.org

Ivona BRAJEVIC
University of Belgrade
Faculty of Mathematics
Studentski trg 16
SERBIA
ivona.brajevic@googlemail.com

*Abstract:* Cuckoo Search is one of the recent swarm itelligence metaheuritics. It has been succesfuly applied to a number of optimization problems, but is stil not very well researched. In this paper we present a parallelized version of the Cuckoo Search algorithm. The parallelization is implemented using CUDA architecture. The algorithm is significantly changed compared to the sequential version. Changes are partialy done to exploit the power of mass parallelization by the graphical processing unit and partialy as a consequence of the memory access restrictions that exist in CUDA. Tests on standard benchmark functions show that our proposed parallized algorithm greatly decreases the execution time and achieves similar or slightly better quality of the results compared to the sequential algorithm.

*Key–Words:* Cuckoo search, Swarm intelligence, Optimization metaheuristics, Parallel algorithms

## 1 Introduction

In recent years a wide range of nature inspired algorithms has been developed for solving hard optimization problems. Among such algorithms swarm intelligence is becoming prominent. Swarm intelligence is collective behavior of decentralized, self-organized systems. A wide range of animal and insect species like fish, birds and ants exhibit this type of behavior where many extremely primitive individuals exhibit remarkable collective intelligence and by doing so greatly increase their chance of survival in nature.

Swarm intelligence has inspired the development of many metaheuristics for solving hard combinatorial as well as continuous optimization problems. They include Ant Colony Optimization [1], Particle Swarm Optimization [2], Artificial Bee Colony Optimization [3] etc. with numerous improvements [4], [5], [6], [7].

A new and very promising member of the swarm intelligence metaheuristics family is the Cuckoo Search (CS) algorithm which mimics the behavior of the brood parasites [8], [9]. It has not yet been thoroughly researched, but there have already been successful applications to many different problems like component design [10], training neural models [11], [12], mesh optimization [13], test data generation [14], etc. There have also been several attempts to improve its performance by adding changes to the basic algorithm [15], [16], [17], [18] or by taking special consideration to the generation of the initial population [19]. One of the most successful modification is the Cuckoo Optimization Method but it has a significant increase in the algorithm complexity [20].

Closely related to swarm intelligence algorithms are different types of older evolutionary algorithms that are also population based. Many of the concepts used to improve performance of evolutionary algorithms can also be applied to algorithms based on swarm intelligence. One of the common problems is transition from global to local search i.e. moving from wide search to a fine one localized near already found good solutions [21], [22]. Another approach to improving the performance of population based algorithms is hybridization that combine more than one of such algorithms [23], [24].

Population based algorithms are very suitable for parallelization. This is due to the fact that such algorithms always contain large number of population members, each conducting very similar tasks. It has been shown that even superlinear speed improvement compared to the sequential version of the algorithm can be achieved [25] by using the island based approach where separate colonies are executed in parallel.

This research was supported by Ministry of education and science of Republic of Serbia, Grants III-44006 and 171006.

Until recently massive parallelization has been reserved for supercomputers only, however nowadays with the development of the powerful Graphic Processor Units (GPU), it become available even on average personal computers. The GPU has evolved into a highly parallel, multi-threaded, many-core processor with tremendous computational power and very high memory bandwidth. Several tools have been created for developing software to exploit the power of the GPU like NVIDIAs CUDA (Compute Unified Device Architecture) [26], Khronos Group's OpenCL (Open Computing Language) [27] and Microsoft DirectCompute which is a part of Microsoft DirectX [28].

In this paper we focus on the parallelization of the Cuckoo Search algorithm on the GPU using CUDA. Previously the parallelization of the CS has been done using multi-core processors [29], but that approach yields much lower number of threads that are executed simultaneously, compared to the possibilities of the GPU.

In our algorithm parallelization is used on three levels. First, parallel reduction is used to speedup the calculation of the fitness function for colony members. Second, all members of one colony are calculated in parallel in one block which contains several threads. Finally, several colonies are run in parallel in separate thread blocks.

The developed algorithm has been designed in a way to comply with CUDA memory access restrictions. We show in our tests that proposed approach greatly increases the speed of calculation giving similar or slightly better quality results compared to the sequential algorithm. Slightly better quality results are achieved even though we only parallelized the basic version of the algorithm and did not introduce any communication among parallel entities, which would be another possibility. The improvement is due to more systematic exploration.

The rest of the paper is organized as follows. In the next section the sequential Cuckoo Search algorithm is presented. The parallel version of this algorithm is presented in the following section. In the final section, experimental results and discussion are presented.

# 2 Cuckoo Search

Cuckoo search (CS) is an optimization algorithm that has been inspired by the brood parasitism of some cuckoo species. Cuckoos lay their eggs in the nests of other host birds (of other species). The shape and color of the cuckoo eggs have evolved to mimic the eggs of the host. If host birds discover that the eggs in the nest are not their own, they will either throw these alien eggs away or abandon their nest and build a new nest elsewhere. If the cuckoo eggs hatch, the cycle is repeated.

This type of behavior has been converted to a meta-heuristic called Cuckoo Search in the following way. Each egg in a nest represents a solution, and cuckoo egg represents a new solution. The idea is to create new, similar and potentially better solutions (cuckoos) to replace the not-so-good solutions in the nests. In the simplest form, each nest contains one egg.

CS is based on three idealized rules:

1. Each cuckoo in the colony lays one egg (solution) at a time, and dumps it in a randomly chosen nest.

2. The best solution will be carried to the next generation.

3. The number of available hosts nests is fixed, and the egg laid by a cuckoo is discovered by the host bird with a probability $p_a$. The discovering (discarding) operation is only done on some set of worst nests.

These rules can be converted to the standard CS algorithm given by the following pseudo-code [8]:

Objective function: $f(X)$, $X = (x_1, x_2, .., x_d)$

Generate an initial population of $n$ host nests;
**while** ($t < MaxGeneration$) or ($stopcriterion$) **do**
  Get a cuckoo randomly (say, $i$) and replace its solution by performing Levy flight;
  Evaluate its quality/fitness $F_i$

  Choose a nest among $n$ nests (say, $j$) randomly;
  **if** $F_i < F_j$ **then**
    Replace $j$ by the new solution;
  **end if**

  A fraction ($p_a$) of the worse nests are abandoned and new ones are built;
  Keep the best solutions/nests;
  Rank the solutions/nests and find the current best;

ISBN: 978-1-61804-158-6

Pass the current best solutions to the next generation;

**end while**

Levy flight, noticed in the nature, which is a combination of short and very long steps and sudden turns is of essential importance for the CS algorithm; it is performed using the following equation:

$$X_i(t+1) = X_i(t) + \alpha \bigotimes Levy(\lambda), \qquad (1)$$

where $\alpha$ ($\alpha > 0$) represents a step size. This step size should be closely related to the scale of the test function that the algorithm is applied on. In most cases, $\alpha$ can be set to the value of 1. The product $\bigotimes$ is used for entry-wise multiplications. It has been shown that the use of Levy flight is much more efficient in exploring the search space as its step length is significantly longer when a large number of steps is performed compared to a simple random walk. The random step length is drawn from a Levy distribution which has an infinite variance with an infinite mean:

$$Levy \sim u = t^{-\lambda} \ , \ \lambda \in (0,3] \qquad (2)$$

The consecutive positions generated through steps/iterations of a cuckoo, create a random walk process which obeys a power-law step length distribution with a heavy tail.

Some of the main advantages of the CS algorithm compared to other population based methods are that it is relatively easy to implement and it has a very small number of parameters that control the method.

# 3 Parallelization of the CS algorithm

In the parallelization of CS we assume that it is used to optimize functions that are similar to standard test functions like the Sphere, Rosenbrocks valley, Schwefels and others. These functions are in the form of sums that makes it possible to greatly increase calculation speed using parallel reduction.

## 3.1 Analysis

When analyzing the sequential CS algorithm we notice two parts of it that are not natural to parallel algorithms.

The first one is to generate one new solution $F_i$ from nest $i$ and compare it to a single solution $F_j$ that corresponds to nest $j$. In the sequential algorithm this is done to minimize the number of objective function evaluations. In the case of parallel algorithm, in which separate threads are dedicated to individual nests, this is not an advantage for the calculation speed. The reason for this is that if the fitness function is calculated for only one nest (thread), during that step all other threads will be idle (a high level of divergence amongst threads).

The second problematic part in the sequential CS is the sorting of solutions corresponding to nests and abandoning the worst $p_a$ fraction. In case of parallel algorithm it is more natural to use a partial parallel reduction for the maximization problem. If parallel reduction is stopped at the third step we know that all the values that have passed to this stage are greater than at least 3 other elements which means they have a high probability of belonging to the worst 25%. This makes them suitable for change using Levy flight as explained in the previous section. It is not of essential importance in the CS algorithm that the worst $p_a$ fraction is abandoned since that stage is used just for diversification of the search. What is important is that the best solution is passed to the next generation and that some "relatively bad" nests are abandoned, which is achieved with this reduction approach.

# 4 Tests and Results

In this section we compare the performance of the sequential and our proposed parallelized CUDA implementation of the CS algorithm. Both algorithms are implemented using Microsoft Visual Studio 2010 combined with CUDA version 4.0 for the parallel implementation. The calculations have been done on a machine with Intel(R) Core(TM) i7-2630 QM CPU @ 2.00 GHz, 4GB of DDR3-1333 RAM, with Nvidia GTX 540M 1GB graphics card running on Microsoft Windows 7 Home Premium 64-bit. The graphics card had 96 CUDA Cores.

Due to the restraints given by the amount of shared memory that a thread block can efficiently use, the block size used in our implementation is 25x16. The optimal size of the block has been calculated using the CUDA occupancy calculator [30]. The block size means that each colony has 25 nests and that 16 threads are dedicated to each nest. In our tests we first compare the quality of results, to verify that the proposed approach does not degrade the performance. In the tests we have used only three of the simpler benchmark functions for which there is no necessity for fine tuning of the algorithm which is common in the improved versions of the CS. We did that since the

ISBN: 978-1-61804-158-6

Table 1: Comparison of quality of results achieved by sequential and parallel CS algorithm.

| Function | Number of | Parallel CS | | | Sequential CS | | |
|---|---|---|---|---|---|---|---|
| | Parameters | Average | Stdev. | Best | Average | Stdev. | Best |
| Sphere | 5 | 3.54E-05 | 2.88E-05 | 4.11E-06 | **9.67e-13** | **1.36e-12** | **2.71e-14** |
| Min=0 | 10 | 0.0055 | 0.0110 | **0.0004** | **5.34e-4** | **0.0021** | 0.0029 |
| | 16 | **0.0383** | **0.0312** | **0.0050** | 0.1356 | 0.0476 | 0.0522 |
| Rastrigin | 5 | **0.7926** | **0.7712** | **0.0011** | 2.8088 | 0.8766 | 0.6566 |
| Min=0 | 10 | **2.787** | **2.069** | **0.111** | 21.962 | 3.896 | 14.583 |
| | 16 | **6.526** | **3.633** | **1.127** | 58.491 | 5.064 | 50.303 |
| Schwefell | 5 | **-2051.39** | **41.52** | **-2092.0** | -1873.1 | 65.1 | -2037.7 |
| Min= | 10 | **-3615.0** | **144.5** | **-4063.5** | -3.094.3 | 162.9 | -3434.1 |
| - | 16 | **-5224.5** | 289.6 | **-5833.6** | -4272.6 | **239.4** | -4730.1 |
| 418.9*NP | | | | | | | |

goal of our proposed approach is to significantly increase the calculation speed and not to try to improve the quality of results compared to the sequential CS algorithm.

The sequential CS algorithm that is used for comparison is presented in the article [8]. The tests are done on the standard test functions Sphere, Rosenbrock's valley, Schwefel's function with 5, 10 and 16 parameters. In Table 1, we present the average and best found solutions as well as standard deviation for 25 independent runs of the algorithm with 10,000 evaluations for each test function. The values for the sequential algorithm are calculated using the Matlab code provided by Yang which has also been used in article [8].

It is noticeable and very interesting that our parallel version of the CS algorithm in most of the test cases outperforms the sequential version when quality of the results is considered. The expected reason for this improvement, which we did not try to achieve since we did not introduce any communication among parallel colonies, is that more diverse solutions are generated and tested. This is the consequence of our modification where lower quality solutions (nests) are not simply overwritten but are used in the creation of new ones. The newly created nests are a linear combination of the good and the lower quality solution, and in a sense represent a transition between them. This has a consequence that in a relatively small number of iterations, the lower quality solution becomes very close to the good one, but the space between them will also be checked. The number of intermediate solutions checked is low enough not to decrease the effectiveness of the basic algorithm, but provides a more systematic exploration of the solution space. The parallel algorithm performs worse in the case of the Sphere function that we believe is due to the fact that in our implementation the step in the Levy's flight does not decrease fast enough.

In the second part of our experiments we analyze the calculation speed of our parallel algorithm and the sequential one. In Table 2, we compare the time needed for the execution of 25 CS algorithm runs when 100,000 function evaluations are conducted for each of the test functions.

For these tests we have implemented the sequential algorithm given by Yang using C++. From the results in Table 2, we can see a tremendous decrease in calculation time of even 10-25 times. The big difference in the level of speedup: 10 times in the case of 5 parameters and 25 times in case of 16 parameters, is due to the use of a fixed block size in all of our tests. This has been done for simplicity of implementation. In the case of smaller problems with only 5 parameters a large number of threads is idle during the program execution. This dramatic speedup is similar to the case of parallelization of the PSO algorithm given in article [31]. This parallelization of PSO using CUDA also involves the use of multiple threads for function evaluation for individual colony members using parallel reduction.

ISBN: 978-1-61804-158-6

Table 2: Comparison of execution speed of the sequential and parallel CS algorithm for 100,000 function evaluations.

| Function | Number of Parameters | Execution time Sequential | Parallel |
|---|---|---|---|
| Sphere | 5 | 15.32 | 1.42 |
| | 10 | 20.52 | 1.42 |
| | 16 | 27.34 | 1.43 |
| Rastrigin | 5 | 17.44 | 1.50 |
| | 10 | 22.21 | 1.50 |
| | 16 | 33.50 | 1.51 |
| Schwefell | 5 | 19.43 | 1.51 |
| | 10 | 24.45 | 1.51 |
| | 16 | 35.67 | 1.52 |

## 5 Conclusion

In this paper we have presented a massive parallel version of the CS. The implementation is done using CUDA for execution on the GPU. The new algorithm consists of parallelization on several levels: first numerous threads are used for evaluating fitness function for individual nests in the colony, all the nests in one colony are executed in parallel and finally several colonies are simulated at the same time. The original sequential algorithm has been significantly modified for this parallel implementation. The proposed parallel algorithm exhibits significant decrease in calculation time of even 25 times compared to the sequential CS algorithm. Since our proposed algorithm has also been able to improve the quality of results for the same number of objective function evaluations even though that was not the target of this research, the future research can include introduction of inter-colony communication that would further improve the quality of the results.

*References:*


[1] M. Dorigo et L. M. Gambardella, Ant Colony System : A Cooperative Learning Approach to the Traveling Salesman Problem, IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1, 53-66 (1997)

[2] J. Kennedy, R. Eberhart, Particle Swarm Optimization. Proceedings of IEEE International Conference on Neural Networks. IV, 1942-1948 (1995)

[3] D. Karaboga, B. Akay, A Comparative Study of Artificial Bee Colony Algorithm, Applied Mathematics and Computation, vol 214, 108-132 (2009)

[4] R. Cheng, M. Yao, Particle Swarm Optimizer with Time-Varying Parameters based on a Novel Operator, Applied Mathematics & Information Sciences, Vol. 5, No. 2 Special Issue: SI, 33-38 (2011)

[5] R. Jovanovic, M. Tuba, An ant colony optimization algorithm with improved pheromone correction strategy for the minimum weight vertex cover problem, Applied Soft Computing, Vol. 11, No 8, 5360-5366 (2011)

[6] R. Jovanovic, M. Tuba: Ant Colony Optimization Algorithm with Pheromone Correction Strategy for Minimum Connected Dominating Set Problem, Computer Science and Information Systems (ComSIS), 9 (4), DOI:10.2298/CSIS110927038J (2012)

[7] I. Brajevic, M. Tuba, An upgraded artificial bee colony algorithm (ABC) for constrained optimization problems, Journal of Intelligent Manufacturing, published Online First, DOI: 10.1007/s10845-011-0621-6 (2012)

[8] Yang, X. S. and Deb, S., Cuckoo search via Levy flights, in: Proc. of World Congress on Nature & Biologically Inspired Computing (NaBIC 2009), 210-214 (2009)

[9] Yang, X. S., and Deb, S. Engineering Optimisation by Cuckoo Search, Int. J. of Mathematical Modelling and Numerical Optimisation, Vol. 1, No. 4, 330-343 (2010)

[10] I. Durgun, A. R. Yildiz, Structural Design Optimization of Vehicle Components Using Cuckoo Search Algorithm, MP Materials Testing, No. 3, 185-188 (2012)

[11] R. A. Vazquez, Training spiking neural models using cuckoo search algorithm, IEEE Congress on Evolutionary Computation (CEC), 679-686 (2011)

[12] Ehsan Valian, Shahram Mohanna and Saeed Tavakoli, Improved Cuckoo Search Algorithm

for Feedforward Neural Network Training, International Journal of Artificial Intelligence & Applications (IJAIA), Vol. 2, No. 3, 36-43 (2011)

[13] S. Walton, O. Hassan, K. Morgan, Reduced order mesh optimisation using proper orthogonal decomposition and a modified cuckoo search, International Journal for Numerical Methods in Engineering, DOI: 10.1002/nme.4400, (2012)

[14] P. R. Srivastava, R. Khandelwal, S. Khandelwal, K. Shobhit, S. S. Sanjay, Automated Test Data Generation Using Cuckoo Search and Tabu Search (CSTS) Algorithm, Journal of Intelligent Systems, Vol. 21, No. 2, 195224 (2012)

[15] N. Bacanin, An object-oriented software implementation of a novel cuckoo search algorithm, Proc. of the 5th European Conference on European Computing Conference (ECC'11), 245-250 (2011)

[16] M. Tuba, M. Subotic, and N. Stanarevic, Modified cuckoo search algorithm for unconstrained optimization problems, Proc. of the 5th European Conference on European Computing Conference (ECC'11), 263-268 (2011)

[17] M. Tuba, M. Subotic, N. Stanarevic, Performance of a modified cuckoo search algorithm for unconstrained optimization problems, WSEAS Transactions on Systems, Vol. 11, No. 2, 62-74 (2012)

[18] S. Walton, O. Hassan, K. Morgan and M. R. Brown, Modified cuckoo search: A new gradient free optimisation algorithm, Chaos, Solitons & Fractals. Vol 44, Is. 9, 710718 (2011)

[19] M. Shatnawi, Starting configuration of Cuckoo Search algorithm using Centroidal Voronoi Tessellations, 11th International Conference on Hybrid Intelligent Systems (HIS), 40-45 (2011)

[20] R. Rajabioun, Cuckoo Optimization Algorithm, Applied Soft Computing, Vol. 11, Is. 8, 5508–5518 (2011)

[21] A. C. Martinez-Estudillo, C. Hervas-Martinez, F. J. Martinez-Estudillo, N. Garcia-Pedrajas, Hybridization of Evolutionary Algorithms and Local Search by Means of a Clustering Method, IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics, Vol. 36, No. 3, 534-545 (2006)

[22] Hecheng Li,Yuping Wang, An Evolutionary Algorithm with Local Search for Convex Quadratic Bilevel Programming Problems, Applied Mathematics & Information Sciences, Vol. 5, No. 2, 139-146 (2011)

[23] G. Jinghui, Rui Shan, A New Method for Modification Consistency of the Judgment Matrix Based on Genetic Ant Algorithm Applied Mathematics & Information Sciences, Vol. 6, Special Issue: 1, 35-39 (2012)

[24] N. Bacanin, M. Tuba, Artificial Bee Colony (ABC) Algorithm for Constrained Optimization Improved with Genetic Operators, Studies in Informatics and Control, Vol 21, Issue 2, 137-146 (2012)

[25] T. Sttzle, Parallelization strategies for Ant Colony Optimization, Parallel Problem Solving from Nature PPSN V, Springer Berlin/Heidelberg, 722-731 (1998)

[26] NVIDIA CUDA C Programming Guide Version 4.0, (2011)

[27] P. O. Jaaskelainen, C. S. de La Lama, P. Huerta, J. H. Takala, OpenCL-based design methodology for application-specific processors, Embedded Computer Systems (SAMOS), 2010 International Conference on, 223-230 (2010)

[28] DirectCompute Lecture Series, http://channel9.msdn.com/tags/DirectCompute-Lecture-Series/, (2011)

[29] M. Subotic, M. Tuba, N. Bacanin, and D. Simian, Parallelized cuckoo search algorithm for unconstrained optimization, Proceedings of the World Congress: Applied Computing Conference (ACC'12), University of Algarve, Faro, Portugal, 151-156 (2012)

[30] CUDA Occupancy Calculator, http:// news.developer.nvidia.com /2007 /03 /cuda _occupancy _.html/, (2007)

[31] L. Mussi, F. Daolio, S. Cagnoni, Evaluation of parallel particle swarm optimization algorithms within the CUDA architecture, Information Sciences, 4642-4657 (2011)