Different approaches in parallelization of the artificial bee colony algorithm

Milos Subotic, Milan Tuba and Nadezda Stanarevic

***Abstract*—**Parallel processing is gaining popularity due to the low cost of multi-core processors. In this paper we propose three different approaches in parallelization of standard artificial bee colony (ABC) algorithm. ABC algorithm was successfully used on many optimization problems, unconstrained and constrained. Our three approaches are independent parallel runs and two variations of multiple swarms parallelization. By using independent parallel runs method we succeeded in achieving faster execution of algorithm since multicore processors can be better utilized. By using multiple swarms technics with some modifications we also obtained better results than the original ABC algorithm. Different types of communications among swarms are proposed and examined. These methods of communication between swarms improved results and allowed adjustments of different ratios between exploration and exploitation. Set of eleven standard benchmark functions was used to test execution speed and quality of results improvements.

***Keywords*—**Artificial bee colony, Metaheuristic optimization, Parallelization, Swarm intelligence, Nature inspired metaheuristic algorithms.

I. Introduction

OPTIMIZATION problems have been solved by many different techniques. As an alternative to the traditional methods in operations research, heuristic methods have been developed. A branch of nature inspired algorithms which are called swarm intelligence is focused on insect behavior in order to develop some metaheuristics which can mimic insect's problem solution abilities. It also has been shown that these algorithms can provide better solutions in comparison to classical algorithms. Artificial bee colony (ABC) algorithm is a relatively new member of swarm intelligence. ABC tries to model natural behavior of real honey bees in food foraging. In ABC system, artificial bees fly around in a multidimensional search space and some (employed and onlooker bees) choose food sources depending on their own experience and also experience of their nest mates, and adjust their positions [1].

In the ABC algorithm, the colony of artificial bees contains three groups of bees: employed bees, onlookers and scouts. The number of employed bees is equal to the number of food sources and an employed bee is assigned to one of the sources. Short pseudo-code of the ABC algorithm is given below [2]:

1. Initialize the population of solutions
2. Evaluate the population
3. Produce new solutions for the employed bees
4. Apply the greedy selection process
5. Calculate the probability values
6. Produce the new solutions for the onlookers
7. Apply the greedy selection process
8. Determine the abandoned solution for the scout, and replace it with a new randomly produced solution
9. Memorize the best solution achieved so far

For every food source, there is only one employed bee.

The ABC algorithm was selected because it is a simple algorithm with few parameters and it has achieved promising results in numerical benchmark optimizations.

Engelbrecht defined the basic components in any optimization problem:

- An objective function which represents the quantity to be optimized, i.e., the quantity to be minimized or maximized.
- A set of unknown variables which affect the value of the objective function, the number of variables defines the dimension of the problem.
- A set of constraints that limit the values that can be assigned to variables. Most of the problems have at least one set of boundary constraints, which define the range of values that each variable can take [3].

A. ABC Algorithm

Although there are several models based on honeybees [4] our work is based on the model initially proposed by Karaboga and Basturk [5] and lately developed by Karaboga and Bastruk [6] that solves numerical optimization problems. An important difference between ABC and other swarm intelligence algorithms is that in the ABC algorithm the possible solutions represent food sources (flowers), not individuals (honeybees). The number of onlooker and employee bees is the same. Onlookers are allocated to a food source based on the profitability. Like the employed bees, onlookers calculate a new solution from its food source. When a food source is depleted, the bee or bees employed on it become unemployed and they have to decide between either becoming a scout bee and finding another food source to exploit randomly or

Manuscript received March 03, 2011.

The research was supported by the Ministry of Science, Republic of Serbia, Project No. III 44006

M. Tuba is with the Faculty of Computer Science, Megatrend University, Belgrade, Serbia, e-mail: tuba@ieee.org

M. Subotic is with the Faculty of Computer Science, Megatrend University, Belgrade, Serbia, e-mail: milos.subotic@gmail.com

N. Stanarevic is with the Faculty of Faculty of Computer Science, Megatrend University, Belgrade, Serbia, e-mail: srna@stanarevic.com

returning to the hive as onlooker bees and waiting for information about other food sources currently exploited. After certain number of cycles, if food source cannot be further improved, it is abandoned and replaced by randomly generated food source. This is called exploration process and it is performed by the third group of bees in colony – scout bees. Hence, employed and onlooker bees carry out exploitation process, while scout bees perform exploration.

II. Benchmark functions

In this paper we used set of eleven well known benchmark functions. Function $f_1(x)$ is Sphere function that is continuous, convex and unimodal. x is in the interval of $[-100, 100]$. Global minimum value for this function is 0 and optimum solution is $x_{opt} =(x_1, x_2, \ldots, x_n) = (0, 0, \ldots , 0)$. Surface plot of $f_1(x)$ is shown in Figure 1.

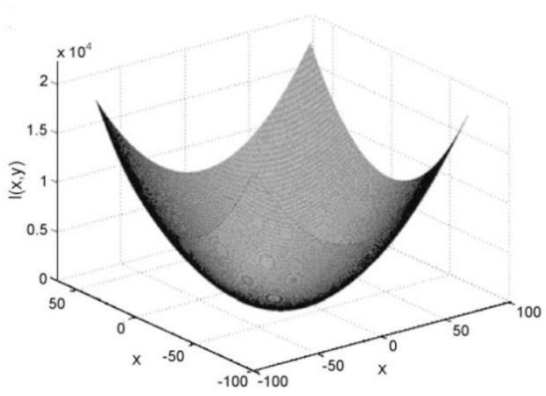

Fig. 1, Sphere function: surface plot

Function $f_2(x)$ is Griewank function. x is in the interval of $[-600, 600]$. The global minimum value for this function is 0 and the corresponding global optimum solution is $x_{opt} =(x_1, x_2, \ldots, x_n) = (100, 100, \ldots , 100)$. Since the number of local optima increases with the dimensionality, this function is strongly multimodal. The multimodality disappears for sufficiently high dimensionalities $(n > 30)$ and the problem seems unimodal. Surface plot of $f_2(x)$ is shown in Figure 2.

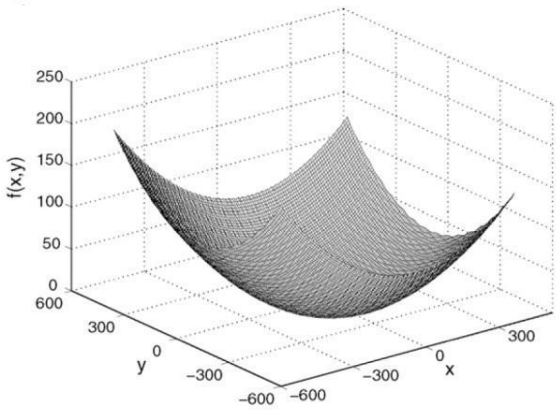

Fig. 2, Griewank function: surface plot

Function $f_3(x)$ is Rastrigin function. This function is based on Sphere function with the addition of cosine modulation to produce many local minima. Thus the function is multimodal. The locations of the minima are regularly distributed. The difficult part about finding optimal solutions to this function is that an optimization algorithm easily can be trapped in a local optimum on its way towards the global optimum. x is in the interval of $[-5.12, 5.12]$. The global minimum value for this function is 0 and the corresponding global optimum solution is $x_{opt} =(x_1, x_2, \ldots, x_n) = (0, 0, \ldots , 0)$. Surface plot of $f_3(x)$ is shown in Figure 3.

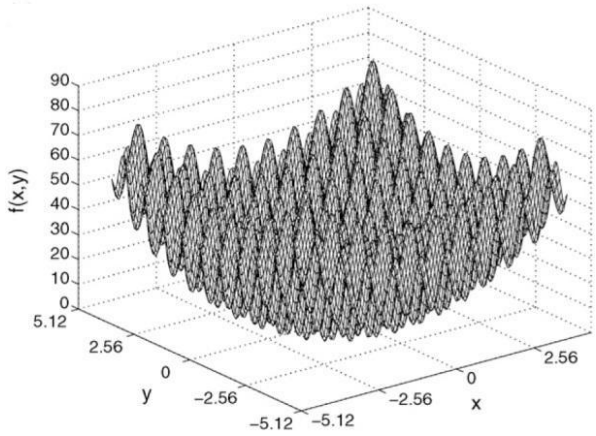

Fig. 3, Rastrigin function: surface plot

Function $f_4(x)$ is well-known classic optimization problem: Rosenbrock valley. The global optimum is inside a long, narrow, parabolic-shaped flat valley. Since it is difficult to converge to the global optimum of this function, the variables are strongly dependent, and the gradients generally do not point towards the optimum, this problem is repeatedly used to test the performance of the optimization algorithms. x is in the interval of $[-50, 50]$. Global minimum value for this function is 0 and optimum solution is $x_{opt} =(x_1, x_2, \ldots, x_n) = (1, 1, \ldots , 1)$. Global optimum is the only optimum, function is unimodal. Surface plot of $f_4(x)$ is shown in Figure 4.

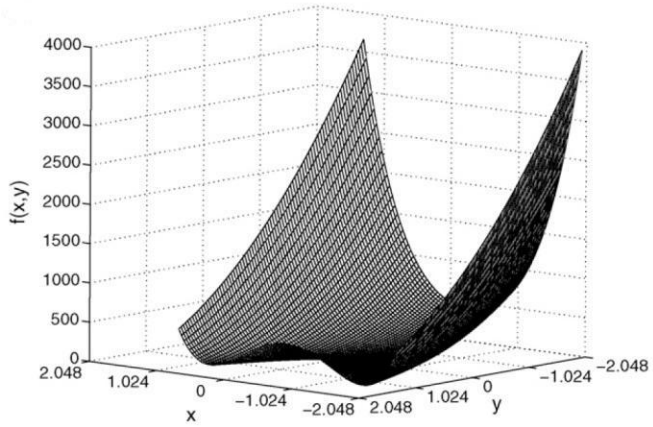

Fig. 4, Rosenbrock function: surface plot

The fifth function $f_5(x)$ is Schwefel function whose value is $f_5(x) = - 418.9829 * n$ at its global minimum $(420.9867, 420.9867, \ldots, 420.9867)$. Schwefel's function is deceptive in that the global minimum is geometrically distant, over the parameter space, from the next best local minima. Therefore,

the search algorithms are potentially prone to convergence in the wrong direction. Test area is usually restricted to hypercube $-500 \le xi \le 500,\ i = 1, \ldots, n$. Its main difficulty is that its gradient is not oriented along their axis due to the epitasis among their variables; in this way, the algorithms that use the gradient converge very slowly. This is a widely used multimodal test function. The surface of Schwefel function is composed of a great number of peaks and valleys. The function has a second best minimum far from the global minimum where many search algorithms are trapped. Moreover, the global minimum is near the bounds of the domain. Surface plot of $f_5(x)$ is shown in Figure 5. Normalized version of this function is used in tests. $f_5(x)=418.9829 * n + \sum_1^n -x_i sin(\sqrt{|x_i|})$ where x is in the interval of $[-500, 500]$.

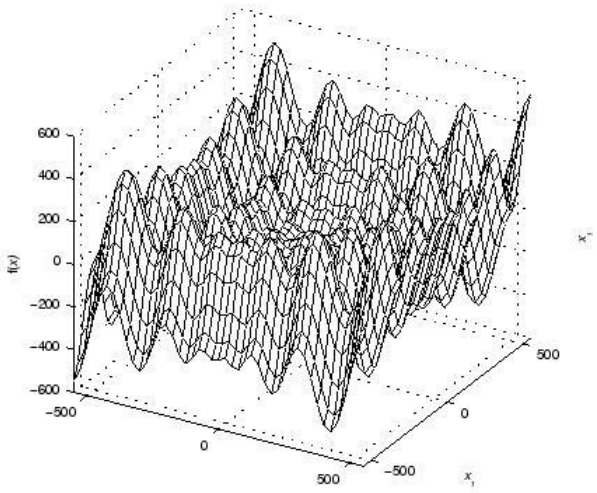

Fig. 5, Schwefel function: surface plot

The sixth test function is Beale function. Usually this function is used with only two parameters. Search domain for Beale function is $-4.5 \le xi \le 4.5$, for $i = 1, 2$. Global minimum is $f_6(3, 0.5) = 0$. Surface plot of $f_6(x)$ is shown in Figure 6.

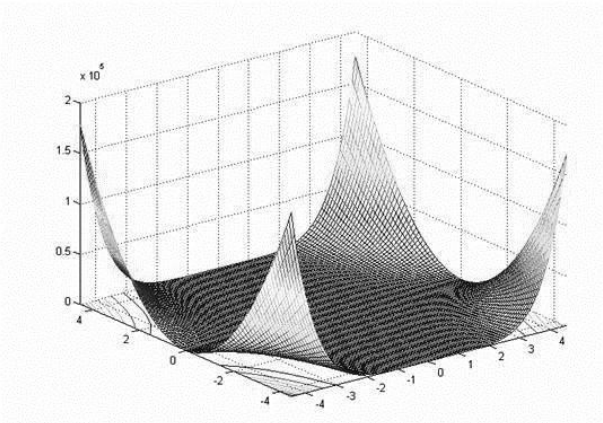

Fig. 6, Surface plot of Beale function

The seventh test function is Booth function. This function has several local minimums. Search domain for Booth function is $-10 \le x_i \le 10,\ i = 1, 2$. Global Minimum is $f_7\ (1, 3) = 0$. Surface plot of $f_7(x)$ is shown in Figure 7.

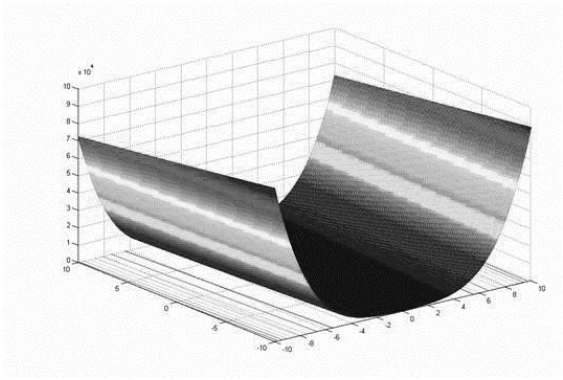

Fig. 7, Surface plot of Booth function

Dixon and Price function is used as eighth test function. The number of parameters is not determinate for this function. Search domain for Dixon and Price function is $-10 \le x_i \le 10$, $i = 1, 2. \ldots n$. Global Minimum is $f_8\ (x) = 0$. Surface plot of $f_8(x)$ is shown in Figure 8.

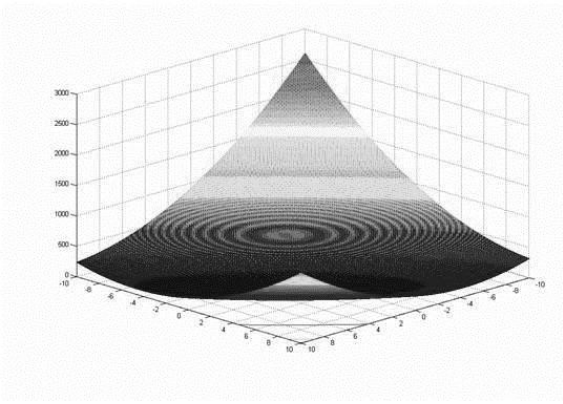

Fig. 8, Surface plot of Dixon and Price function

The ninth test function is Matyas function. This function has only one minimum, the global one, and has no other local minimums. This function has two parameters. Search space for this function is $-10 \le x_i \le 10,\ i = 1, 2$. The global minimum for this function is $f_9(0, 0) = 0$. Surface plot of $f_9(x)$ is shown in Figure 9.

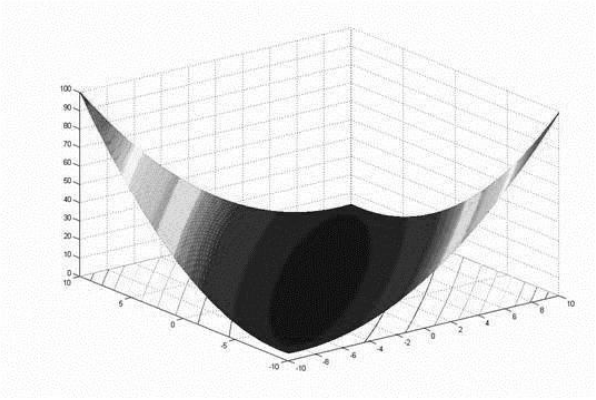

Fig. 9, Surface plot of Matyas function

Function $f_{10}(x)$ is step function. It is a discontinuous function and has one minimum. This function represents the problem of flat surfaces. It is very hard for algorithms without variable step sizes to conquer flat surfaces problems because there is

no information about which direction can provide optimal solution. Surface plot is shown in Fig. 10.

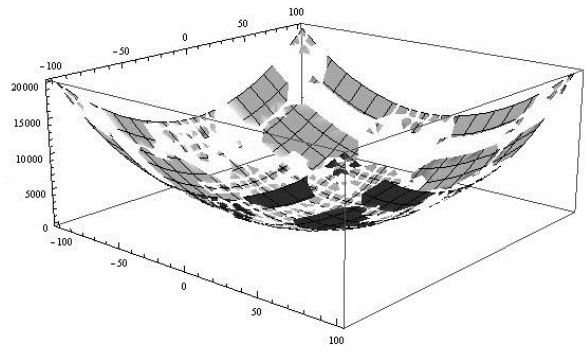

Fig. 10, Surface plot of step function

Our eleventh function $f_{11}(x)$ used in our benchmark set is Ackley function. The Ackley Function is a continuous, multimodal function obtained by modulating an exponential function with a cosine wave of moderate amplitude. Originally, it was formulated by Ackley only for the two – dimensional case; it is presented here in a generalized, scalable version. Its topology is characterized by an almost flat (due to the dominating exponential) outer region and a central hole or peak where the modulations by the cosine wave become more and more influential. The global minimum value for this function is *0* and the corresponding global optimum solution is $x_{opt} = (x_1, x_2, \ldots, x_n) = (0, 0, \ldots, 0)$.Surface plot of $f_{11}(x)$ is shown in Fig. 11. X is in the interval of *[-32, 32]*.

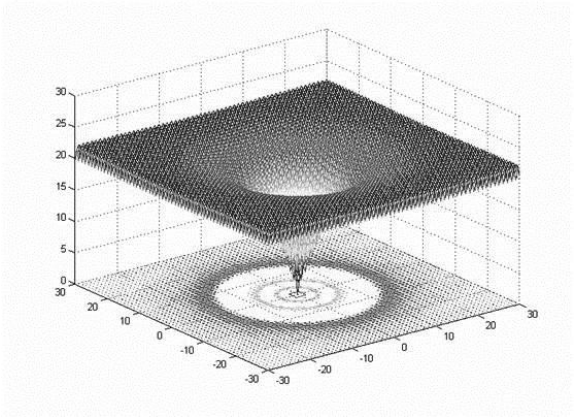

Fig. 11, Surface plot of Ackley function

In experiments, $f_1(x)$, Sphere function has 5 parameters, $f_2(x)$ Griewank, $f_3(x)$ Rastrigin, $f_4(x)$ Rosenbrock, $f_5(x)$ Schwefel, $f_8(x)$ Dixon and Price, $f_{10}(x)$ Step and $f_{11}(x)$ Ackley functions have 50 parameters. Functions $f_6(x)$ Beale, $f_7(x)$ Booth and $f_9(x)$ Matyas have 2 parameters. Number of parameters, parameter ranges, formulations and global optimum values of these functions are given in Table 1.

TABLE I
NUMERICAL BENCHMARK FUNCTIONS

Name	Function	No. of param.	Ranges	Min
Sphere	$f_1(x)=\sum_{i=1}^{n} X_i^2$	5	-100 ≤ xi ≤ 100	0
Griewank	$f_2(x) = \sum_{i=1}^{n} \frac{X_i^2}{4000} - \prod_{i=1}^{n} \cos(x_i/\sqrt{i}) + 1$	50	-600 ≤ xi ≤ 600	0
Rastrigin	$f_3(x) = 10n + \sum_{i=1}^{n}(X_i^2 - 10\cos(2\pi X_i))$	50	-5.12 ≤ xi ≤ 5.12	0
Rosenbrock	$f_4(x) = \sum_{i=1}^{n-1}\left[100(x_i^2 - X_{i+1})^2 + (X_i - 1)^2\right]$	50	-50 ≤ xi ≤ 50	0
Schwefel	$f_5(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{\lvert x_i \rvert})$	50	– 500 ≤ xi ≤ 500	418.9829*n
Beale	$f_6(x) = (1.5 - x(1-y))^2 + (2.25 - x(1-y^2))^2 + (2.625 - x(1-y^3))^2$	2	$-4.5 \le x_i \le 4.5$	0
Booth	$f_7(x)=(x_1+2x_2-7)^2 + (2x_1+x_2-5)^2$	2	$-10 \le x_i \le 10$	0
Dixon and Price	$f_8(x)=\sum_{i=1}^{n} i(2x_i^2 - x_{i-1})^2 + (x_1+1)^2$	50	$-10 \le x_i \le 10$	0
Matyas	$f_9(x)= 0.26\,(x_1^2 + x_2^2) - 0.48x_1x_2$	2	$-10 \le x_i \le 10$	0
step	$f_{10} = \sum_{i=1}^{n} (\lvert x_i + 0.5 \rvert)^2$	50	$-100 \le x_i \le 100$	0
Ackley	$f_{11}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$	50	$-32 \le x_i \le 32$	0

III. Different parallelization approaches

We are witnessing a dramatic change in computer architecture due to the multicore paradigm shift, as every electronic device from cell phones to supercomputers confronts parallelism of unprecedented scale [7]. Majority of processors today have multiple cores and even for a single core multiple treads can be implemented. In general, a system of *n* parallel processors, each of speed *k*, is less efficient than one processor of speed *n * k*. However, the parallel system is usually much cheaper to build and its power consumption is significantly smaller. Another issue is the memory wall. Processor clock rates have been increasing much faster for some time than memory clock rates, and that trend is continuing. Increasing the Gigahertz rates on microprocessors did not improve performance significantly as it did in the past, and larger and larger caches and other things that help alleviate some of the issues — but not all of them — on memory were increasingly added. So, multicore processors actually address this issue by slowing the increase in clock rate on the processor so that the problem doesn't get worse. To that end research in parallelization is of great importance. Parallelization of algorithms has proven to be very powerful method in the case of population based algorithms like ant colony optimization (ACO) and genetic algorithms [8]. Thus the aim of this work was to examine implementation of parallelization on ABC algorithm using three different approaches.

The main question in implementing parallelization is the level of parallelization. The most common solution is creating every cycle in ABC algorithm as an independent thread. This is too fine grained implementation, and it has one major disadvantage. There is a rather small portion of work in each cycle, so extensive use of CPU time for creating threads and their synchronizations exceeds the benefits of parallel execution of each cycle. ABC algorithm contains thousands of cycles. Creating and synchronizing such large number of threads can be slower by far than using a serial execution of cycles. For that reason, we didn't use this kind of implementation, but implemented three other approaches.

Our approaches are:

1. Parallel independent runs
2. Multiple swarms – one best solution
3. Multiple swarms – best solutions from all swarms

Increasing performance is the main focus of parallel independent runs approach. Multiple swarm approaches aim at getting better results.

A. Independent parallel runs approach

It is desirable to run population based heuristics many times, because they do not provide exact result but rather give approximation as final result. It is quite useful to run all iterations simultaneously in order to save time. In this approach threads have no communication between themselves at all. Every thread runs the same sequential ABC algorithm with different random seeds. The final solution is the best one of all the independent runs. The speed increases almost as many folds as there are execution cores in system. Independent parallel runs approach is too coursed grained and there are no speed gains for one single runtime. On single execution core system this implementation can be slower than serial execution of all runs. This can be explained by high cost of switching CPU between threads. But for today's modern CPU's that is not an issue, hence almost every PC has a at list processor with two cores.

B. Multiple swarms approaches

The other two approaches we used are based on multiple swarm tactics. The idea was to use more than one swarm on the same search space. These swarms are able to communicate with each other in order to exchange the results. After every communication the new solution matrix is formed in all swarms, based on best-so-far solutions from each swarm. Multiple swarms can find more useful solutions and narrow the search space. Trapping in local optimum can be avoided by using numerous swarms. One of the questions referring this method is how many swarms should be used? It is suggested that the number of swarms should be equal to the number of rows in solution matrix. That is a half of the colony size. Our experiments imply that the best results can be achieved in this way. The other question is how often the swarms should communicate. The period between two communications can be determined by the number of cycles or by the time unit. In our experience, it is better to use number of cycles then time unit. Since on different systems, various amounts of computational work can be done in the same time, only a few communications can occur on the faster system, while during the same algorithm execution on the slower system, number of communications can be significantly greater. In our experiments the number of cycles between two communications was determined by dividing total number of cycles by the number of swarms. After certain number of cycles, every swarm sends its best-so-far solution to all other swarms. The key difference between "multiple swarms – one best solution" and "multiple swarms – best solutions from all swarms" approaches is in the role of solutions obtained from each swarm in formation of new solution matrix after every communication.

In the "multiple swarms – one best solution" approach, the best solution from every single swarm is collected after certain number of cycles. Then, the best of all collected solutions is sent back to each swarm to replace one of the solutions in the existing solution matrix. "Multiple swarms – best solutions from all swarms" uses another (different) strategy. Namely, every swarm replaces its solution matrix with the new one which is formed after certain number of cycles. Every row of the new solution matrix represents the best solution from one of the swarms.

IV. Test results

All of the parallelization approaches have been implemented using Java programming language. The Java platform is designed from the ground up to support concurrent

programming, with basic concurrency support in the Java programming language and the Java class libraries. In the Java programming language, concurrent programming is mostly concerned with threads. Threads are sometimes called lightweight processes. Both processes and threads provide an execution environment, but creating a new thread requires fewer resources than creating a new process. Each thread is associated with an instance of the class Thread. A thread is a unit of processing in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently. For test purposes, we created test application in Java programming language based on Karaboga`s and Bastuk`s software in C programming language [2], [6], [9]. All of our tests have been performed on an Intel(R) Q6600 @ 3.0 GHz with 4 GB of RAM with Microsoft Windows XP Professional Edition Version 2003 Service Pack 3. We used Sun Microsystems Java Virtual Machine, and NetBeans as IDE.

In the ABC algorithm there are only three parameters to be modified: number of solutions, total number of iterations (cycles) and abandonment limit. Number of solutions (*SN*) represents the total number of solutions as well as the number of employer bees and number of onlooker bees. The colony size is *2 * SN*. Total number of iterations (*MCN*) represents max number of cycles. Test parameters for all benchmark function are given in table 2. Limits are calculated by formula: *limit = 0.25 x NP x D* [10].

TABLE II
PARAMETERS USED IN ABC ALGORITHM

Function	Max cycle	NP	Runs
Sphere (f_1)	2000	20	30
Griewank (f_2)	2000	20	30
Rastrigin (f_3)	2000	20	30
Rosenbrock (f_4)	2000	20	30
Schwefel (f_5)	2000	20	30
Beale (f_6)	2000	20	30
Booth (f_7)	2000	20	30
Dixon & Price (f_8)	2000	20	30
Matyas (f_9)	2000	20	30
step (f_{10})	2000	20	30
Ackley (f_{11})	2000	20	30

First test is a speed test. It demonstrates a speed gains when all runs are executing in the parallel manner. It is clear that if function is complicated, improvements are more obvious. In ideal case, parallel independent runs approach should be four time faster that serial runs, hence our test PC has CPU with four physical cores. Speed test results are shown in Table 3. Times are given in seconds. As we can see from result table, for some function, the ratio between parallel independent runs approach and serial run approach is almost four. That is the case with Griewank, Rastrigin, Rosenbrock, Schwefel, step and Ackley functions. Dixon and Price function shows less gains from running parallel. Sphere, Beale, Booth and Matyas functions are slower when they are running in the parallel manner.

TABLE III
SPEED TESTS FOR BENCHMARK FUNCTIONS

Function	Serial runs	Parallel indep. runs
Sphere (f_1)	**2.4**	14.7
Griewank (f_2)	39.2	**11.2**
Rastrigin (f_3)	32.2	**9.1**
Rosenbrock (f_4)	34.3	**12.3**
Schwefel (f_5)	32.9	**9.8**
Beale (f_6)	**3.1**	12.9
Booth (f_7)	**3.0**	12.8
Dixon & Price (f_8)	17.4	**7.2**
Matyas (f_9)	**2.5**	13.0
step (f_{10})	23.3	**8.7**
Ackley (f_{11})	41.4	**13.3**

Since these objective functions can be calculated quickly, they require small amount of CPU time when serial runs are used. More CPU time is used for creating and synchronizing threads then for calculating objective function. A sphere function is very simple function, so calculating objective functions is not CPU challenging. It is similar for Beale, Booth and Matyas functions. These functions have only two parameters, thus they have low demanding for CPU time, and parallel independent runs approach is slower than serial runs approach. Computational time can be prolonged by increasing the number of parameters. Independent parallel runs approach can achieve better results than serial runs with greater number of parameters. Speed tests for Sphere function with various numbers of parameters are shown in Table 4. Time is expressed in seconds.

TABLE IV
DIFFERENT NUMBER OF PARAMETERS FOR SPHERE FUNCTION

Number of parameters (D)	Serial runs	Parallel independent runs
5	**2.4**	14.7
50	**4.1**	15.8
250	12.3	12.3
500	21.5	**8.6**
1000	41.1	**11.6**

Independent parallel runs approach has no influence on quality of results. Approximately the same results are obtained by using both approaches, serial runs and parallel independent runs. Hence we didn't compare results obtained from these two approaches. Independent parallel runs approach is presented as a technic that can be used for speed improvements, not as a technic for better results.

Results from multiple swarm approaches are shown in Table 5.

TABLE V
RESULTS OF TESTED FUNCTIONS

f	Value	Serial Runs	MS-one best sol.	MS-best sol. from all swarms
f_1	Mean	8.003E-05	9.187E-13	**5.876E-14**
	Best	7.640E-06	6.107-14	**3.781E-15**
	Worst	3.356E-04	6.394E-12	**2.497E-13**
	St. dev.	8.203E-05	5.839E-13	**1.361E-13**
f_2	Mean	2.314E-06	2.781E-14	**0**
	Best	2.065E-07	5.794-15	**0**
	Worst	1.215E-05	5.239E-13	**0**
	St. dev.	2.623E-06	4.178E-14	**0**
f_3	Mean	2.170E+01	1.639E-04	**5.866E-05**
	Best	1.218E+01	0	**0**
	Worst	3.146E+01	4.716E-03	**3.964E-04**
	St. dev.	4.652E+00	8.602E-04	**7.369E-04**
f_4	Mean	3.342E+08	1.772E+00	**8.750E-01**
	Best	1.846E+07	6.730E-02	**3.833E-02**
	Worst	8.828E+08	8.088E+00	**1.050E+00**
	St. dev.	2.515E+08	1.794E+00	**1.290E+00**
f_5	Mean	7.774E+02	2.143E+01	**1.560E+01**
	Best	2.993E+02	**5.093E-11**	4.355E-09
	Worst	1.303E+03	2.369E+02	**6.742E+01**
	St. dev.	2.355E+02	5.850E+01	**2.651E+01**
f_6	Mean	2.553E-05	4.540E-09	**9.811E-15**
	Best	2.467E-15	**0**	**0**
	Worst	7.625E-04	9.796E-07	**7.876E-13**
	St. dev.	1.392E-04	1.811E-09	**5.789E-14**
f_7	Mean	0	0	0
	Best	0	0	0
	Worst	0	0	0
	St. dev.	0	0	0
f_8	Mean	3.193E-03	1.437E-05	**3.734E-07**
	Best	1.385E-04	1.172E-07	**5.914E-10**
	Worst	3.339E-02	2.188E-04	**4.790E-5**
	St. dev.	6.003E-03	4.031E-05	**7.609E-06**
f_9	Mean	0	0	0
	Best	0	0	0
	Worst	0	0	0
	St. dev.	0	0	0
f_{10}	Mean	3.928E-07	6.008E-10	**6.937E-13**
	Best	5.754E-08	1.975E-12	**6.297E-14**
	Worst	4.901E-06	6.904E-9	**1.297E-12**
	St. dev.	9.378E-07	1.789E-10	**6.987E-13**
f_{11}	Mean	2.927E-09	7.987E-11	**2.097E-12**
	Best	4.908E-10	3.165E-13	**5.536E-14**
	Worst	9.876E-09	1.290E-10	**7.981E-11**
	St. dev.	1.223E-10	6.084E-11	**7.093E-12**

The quality of results obtained by serial runs, "multiple swarm – one best solution" approach (MS – one best sol.) and "multiple swarm – best solutions from all swarms" (MS – best sol. from all swarms) are compared. Mean solution, best solution, worst solution and standard deviation obtained from 30 runs are observed in Table 5. In order to make the comparison clearer, values below E-15 were assumed to be 0. As we can see, both multiple swarm approaches (one best solution and best solutions from all swarms) achieves better results than serial runs. "Multiple swarms – best solution from all swarms" approach gives slightly better results than "multiple swarms – one best solution" approach. For most test functions, improvements in results quality are significant. All tested approaches obtain zeros for mean, best, worst solution and standard deviation for functions $f_7(x)$ (Booth) and $f_9(x)$ (Matyas). It is due to small number of parameters and fairly small CPU time demands of these functions.

V. CONCLUSIONS

In this paper three different approaches in parallelization of artificial bee colony algorithm ware implemented. The aim was to achieve speed gains by using independent parallel runs approach and to obtain better results by using multiple swarms approaches. Independent parallel runs implementation is significantly faster than serial runs method, especially if objective functions is complicated and demands a lot of CPU time or /and has a great number of parameters. In the future, as the number of execution cores increases, the time difference between parallel and serial runs will be even greater. More precise results can be accomplished using multiple swarms approaches. As shown in Table 5, the results obtained by serial runs method cannot match in terms of quality and consistency the results achieved by multiple swarms methods. Future work will include further investigation of the parallel implementations of ABC algorithm and application to constrained benchmark functions [11] and other real life problems [12]. Constrained problems are more CPU demanding due to the necessity of calculating constrains, so we expect greater speed gains. There are several issues that remain for future work, such as to examine a convergence speed of multiple swarms approaches, and testing all three parallel implementations of artificial bee colony algorithm on more test function with greater number of parameters.

REFERENCES

[1] Jiann-Horng L., Meei-Ru L., Li-Ren H.: A novel bee swarm optimization algorithm with chaotic sequence and psychology model of emotion, Proceedings of the 9th WSEAS International Conference on Systems Theory and Scientific Computation 2009, pp. 87-92.

[2] Karaboga D, Basturk B, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J. of Global Optimization, Volume 39, No. 0925-5001, 2007, pp. 459-471.

[3] Engelbrecht A P, Fundamentals of Computational Swarm Intelligence, chapter 1 Introduction, pages 1–4. Wiley and Sons, 2005

[4] Baykasoglu A, Ozbakır L, Tapkan P, Artificial Bee Colony Algorithm and Its Application to Generalized Assignment Problem, Swarm Intelligence: Focus on Ant and Particle Swarm Optimization, I-Tech Education and Publishing, ISBN: 978-3-902613-09-7, 2007, pages 532, pp. 113-144

[5] Karaboga D., An idea based on honey bee swarm for numerical optimization, Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, Kayseri, Turkey, 2005, http://mf.erciyes.edu.tr/abc/publ.htm
[6] Basturk B., Karaboga D., An artificial bee colony (ABC) algorithm for numeric function optimization, Applied Soft Computing, 2008, Vol. 8, Issue 1, pp. 687-697
[7] Williams S, Oliker L, Vuduc R, Shalf J, Yelick K, Demmel J, Optimization of sparse matrix–vector multiplication on emerging multicore platforms, Parallel Computing, 2009, Volume 35, Issue 3, Pages 178-194
[8] Tanese R., Parallel genetic algorithms for a hypercube, Proceedings of the second international conference on Genetic Algorithms and their Applications, Hillsdale, NJ, Lawrence Erlbaum Associates, Inc. 1987, pp. 177–183
[9] Karaboga D, Basturk B, On the performance of artificial bee colony (ABC) algorithm, Applied Soft Computing, 2007, Volume 8, No. 1568-4946, pp. 687-697
[10] Karaboga D, Akay B, Ozturk C, Artificial Bee Colony (ABC) Optimization Algorithm for Training Feed-Forward Neural Networks, Modeling Decisions for Artificial Intelligence, 2007, Volume 4617/2007, No. 0302-9743, pp. 318-329
[11] D. Karaboga, B. Basturk, Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems, LNCS: Advances in Soft Computing: Foundations of Fuzzy Logic and Soft Computing, 2007, pp. 789–798
[12] Akay B., Karaboga D., Artificial bee colony algorithm for large-scale problems and engineering design optimization, Journal of Intelligent Manufacturing, DOI: 10.1007/s10845-010-0393-4, 2010, pp. 1-14

Milan Tuba received B.S. in mathematics, M.S. in mathematics, M.S. in computer Science, M.Ph. in computer science, Ph.D. in computer science from University of Belgrade and New York University.

From 1983 to 1994 he was in the U.S.A. first as a graduate student and teaching and research assistant at Vanderbilt University in Nashville and Courant Institute of Mathematical Sciences, New York University and later as an assistant professor of electrical engineering at Cooper Union Graduate School of Engineering, New York. During that time he was the founder and director of Microprocessor Lab and VLSI Lab, leader of scientific projects and supervisor of many theses. From 1994 he was associate professor of computer science and Director of Computer Center at University of Belgrade, Faculty of Mathematics, and from 2004 also a Professor of Computer Science and Dean of the College of Computer Science, Megatrend University Belgrade. He was teaching more than 20 graduate and undergraduate courses, from VLSI design and Computer architecture to Computer networks, Operating systems, Image processing, Calculus and Queuing theory. His research interest includes mathematical, queuing theory and heuristic optimizations applied to computer networks, image processing and combinatorial problems. He is the author of more than 100 scientific papers and a monograph. He is coeditor or member of the editorial board or scientific committee of number of scientific journals and conferences.

Prof. Tuba is member of the ACM since 1983, IEEE 1984, New York Academy of Sciences 1987, AMS 1995, SIAM 2009. He participated in many WSEAS Conferences with plenary lectures and articles in Proceedings and Transactions.

Milos Subotic received B.S. in computer science in 2010 from Advanced School of Electrical and Computer Engineering, Belgrade, Serbia and also B.S. in economics in 2006 from Megatrend University of Belgrade.

He is currently Ph.D. student at Faculty of Mathematics, Computer science department, University of Belgrade and works as teaching assistant at Faculty of Computer Science, Megatrend University of Belgrade. He is the coauthor of two papers. His current research interest includes nature inspired metaheuristics.

Mr. Subotic participated in WSEAS conferences.

Nadezda Stanarevic received B.S. in mathematics in 2006 and M.S. in mathematics in 2008 from University of Belgrade, Faculty of Mathematics.

She is currently Ph.D. student at Faculty of Mathematics, Computer science department, University of Belgrade and works as teaching assistant at College of Business, Economy and Entrepreneurship in Belgrade. She is the coauthor of two papers. Her current research interest includes nature inspired metaheuristics.

Ms. Stanarevic participated in WSEAS conferences.