ISSN 1330-3651 (Print), ISSN 1848-6339 (Online) https://doi.org/10.17559/TV-20191217030336
Original scientific paper

Parallel Acceleration and Improvement of Gravitational Field Optimization Algorithm

Lan HUANG, Wen-Xuan WU, Xue-Mei HU*, Sen YANG, Yu LIN, Yan WANG

Abstract: The Gravitational Field Algorithm, a modern optimization algorithm, mainly simulates celestial mechanics and is derived from the Solar Nebular Disk Model (SNDM). It simulates the process of planetary formation to search for the optimal solution. Although this optimization algorithm has more advantages than other optimization algorithms in multi-peak optimization problems, it still has the shortcoming of long computation time when dealing with large-scale datasets or solving complex problems. Therefore, it is necessary to improve the efficiency of the Gravitational Field Algorithm (GFA). In this paper, an optimization method based on multi-population parallel is proposed to accelerate the Gravitational Field Algorithm. With the help of the parallel mechanism in MATLAB, the algorithm execution speed will be improved by using the parallel computing mode of multi-core CPU. In addition, this paper also improves the absorption operation strategy. By comparing the experimental results of eight classical unconstrained optimization problems, it is shown that the computational efficiency of this method is improved compared with the original Gravitational Field Algorithm, and the algorithm accuracy has also been slightly improved.

Keywords: Gravitational Field Algorithm; multi-population; parallel computing

1 INTRODUCTION

Nowadays, optimization problems are quite common in many areas of the real world. The applications of optimization theories and methods have greatly promoted the development of agriculture, transportation, engineering, biotechnology and other research fields [1]. The modern optimization algorithms are widely applied, such as logistics scheduling problem [2], production scheduling optimization problem [3] and the predictive gene control network in bioinformation [4].

Modern optimization algorithms [5] emerged as the times require, which are heuristic algorithms [6-7] rising in the early 1980s. They are based on objective natural phenomena and can solve practical optimization problems by imitating the phenomena or behaviour of nature. For example, Genetic Algorithm (GA) was proposed by John Henry Holland in 1975 [8], which simulated the selection process of natural evolution. Dorigo proposed an ant colony optimization algorithm in 1992 based on the behaviour of ants discovering food and communicating with the members in colony [9]. The Particle Swarm Optimization (PSO) algorithm was proposed by Eberhart and Kennedy in 1995 [10], which simulated a simplified social pattern and behaviour of birds. These optimization algorithms are constantly being improved to cope with a wide variety of applications. However, they also all have their own inherent defects. For example, GA is limited by local optimal solution and genetic bleaching [11], SA algorithm usually has much long running time [12] and PSO algorithm has the problem of low accuracy and is easy to fall into local optimum. To deal with these shortcomings, the Gravitational Field Algorithm as a new simple and effective heuristic search algorithm was proposed by Zheng in 2012 [13]. Compared with other optimization algorithms [13-15], GFA can not only solve the multi-peak problems efficiently, but also converge to global optimum solution soon.

Although GFA is a simple and effective algorithm, it still has a long running time when initial dust population or scale of given problems is large. In order to improve the efficiency of GFA, the parallelization of this algorithm is studied.

Based on the original GFA, Parallel Gravitational Field Algorithm (PGFA) is proposed in this paper. PGFA adopts the strategy that divides initial dust population into multiple subgroups. And these multiple subgroups are allocated to multi-core CPU processors for parallel computation, based on the parallel mechanism in MATLAB. The subgroups in each processor are dealt with independently, and the results in each processor will be summarized to obtain the final optimal solution when all tasks in processors have been completed. Moreover, this paper improves the absorption operation, based on the original GFA. This improvement of absorption operation optimizes the search space for iteration, which can avoid excessive redundant search operations. And it also helps accelerate the iteration process in algorithm.

To verify the performance of PGFA, we test PGFA and original GFA on several common benchmark functions to compare the efficiency and accuracy. The experimental results on eight complex unconstrained problems indicate that PGFA has better accuracy and efficiency than original GFA.

2 ORIGINAL GRAVITATIONAL FIELDALGORITHM

The original Gravitation Field Algorithm is a heuristic search algorithm proposed by Zheng et al. [8]. The idea of this optimization algorithm is derived from the theory of planetary formation which is accepted by most astronomers: the Solar Nebular Disk Model (SNDM) [16].

The main idea of SNDM model is that dust and nebulae originally drift in the universe, and the gravitational field unites them together, eventually forming the planets in solar system. The GFA simulates these natural phenomena to solve optimization problems. All individuals, regarded as the dust population in GFA, represent all the feasible solutions. The mass of dust is calculated according to the objective function, and it will serve as a criterion for evaluating the quality of dust. Each dust attracts each other by gravitational field and eventually gathers at one point. That final point's location represents the optimal solution.

In the original GFA [17], the dusts are initialized randomly, and the flow chart of GFA is shown in Fig. 1. The details of GFA are summarized as follows.

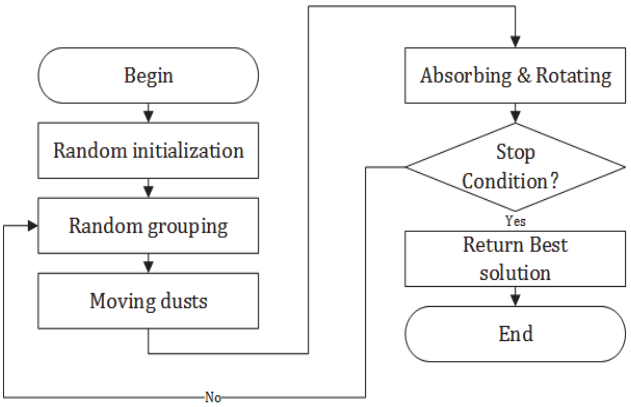

Figure 1 The flow chart of GFA

- First, n dusts particles are initialized randomly. $d_i(i = 1, 2, ..., n)$ represents the feasible solution of a certain problem. d_i can be a scalar or vector. For the j-th dimension space of the i-th dust particle (feasible solution) d_i, the feasible range is set as $\left[X_{jbegin}, X_{jend}\right]$.
- Second, the whole solution space is divided into several subspaces randomly, every subspace is called "group". The dusts with the largest mass value in each group are called center dusts, and the other dusts are called surrounding dusts.
- Third, moving dusts. In GFA, the movement is unidirectional, that is, center dust will not move, and surrounding environmental dust will move toward center dust under the gravity of center dust. The pace of movement is determined by Eq. (1).

$$Pace_i = w \times dis_i \tag{1}$$

where dis_i represents the Euclidean distance between the surrounding dust i and its center [13] dust. And w represents the pace of moving distance [17]. The w = 0.0618 in [13].

- Fourth, absorbing dusts. When the distance between surrounding dusts and their center is less than the threshold set, surrounding dust will be absorbed by center dust. The absorption strategy adopted in [19] is to delete these surrounding dusts directly.
- Fifth, the rotation operation. The aim of Rotation Operator in [14] is to push surrounding dust away from center dust. The direction of departure is not original forward direction, but the random distances in every dimension. To ensure the surrounding dusts will not be pushed too far away, it needs to set a maximum distance that can be pushed out. That is defined by Eq. (2) in [17].

$$withdraw_{\max} = 2 \times dis \times 0.0618 \tag{2}$$

- Sixth, checking whether the termination conditions are met. If one of the algorithm termination conditions is satisfied, the algorithm will be terminated. Otherwise, go to the third step and continue to search.
- According to the detailed description above, the pseudo-code of GFA is given in Algorithm 1.

Algorithm 1 GFA

```
initialdusts ← Initialize()
[groupdusts, center] ← Group(initialdusts)
dusts ← groupdusts
while the stop condition is not met do
    movedusts ← Move(dusts, center); // Moving dust
    absorbdusts ← Absorb(movedusts); //
    rotatedusts ← Rotate(absorbdusts);
dusts ← rotatedusts
end
return best solutions
```

It is reported that original GFA is very effective for solving the approximate optimal solution of unconstrained optimization problems [13, 17]. However, regardless of efficiency or accuracy, it may be difficult to obtain a satisfactory solution for the high-dimensional optimization problems [17]. So it is significant and essential to develop and study the parallel model of GFA.

3 PARALLEL GFA BASED ON PARALLEL MECHANISM IN MATLAB

3.1 Study on the Parallelism of GFA

The parallelism of evolutionary algorithms, based on population model, such as GA and SA, are mainly due to the independence of individual behaviour. Two ideas are mostly adopted in parallel implementation of these evolutionary algorithms. One decomposes the whole algorithm into tasks and executes multiple tasks in parallel. For example, reference [18] proposed a parallel scheme that applied this idea for SA. The other one, such as reference [19] uses the divide-and-conquer model to divide the whole population into several sub-groups and optimize them separately.

GFA is similar to other evolutionary algorithms, such as PSO and GA, as the fact that they are all based on the concept of colony, so GFA can also be paralleled [20]. According to the characteristic of GFA, there are two aspects that have the possibility of being paralleled [21]:

a) The calculation and update of position and mass for individuals in GFA can be paralleled since the behaviours of all dusts are independent, except the center.

b) The process of obtaining and updating the center dusts also can be paralleled. After calculating the position and mass of an individual, the individual will immediately compare with the current center dust mass and update center dust.

GFA has the operation of decomposition solution space, and each iteration of the dust particles is independent. Since the characteristic of GFA do coincide with above parallel pattern, this paper adopts divide-and-conquer [22] method to realize the Parallel Gravitational Field Algorithm (PGFA). This paper divides the whole dust population into several subgroups, and searches the optimal solution independently in parallel. Because of the above process, the search time of PGFA will be reduced greatly. In this paper, PGFA is designed and implemented by the parallel mechanism of MATLAB.

3.2 Introduction of Parallel Computing Platform in MATLAB

As a powerful enterprise-level mathematical software, MATLAB provides us with an efficient and convenient parallel computing architecture. It mainly relies on two tools: Parallel Computing Toolbox (PCT) and Matlab Distributed Computing Server (MDCS) [23]. The two tools support parallel computing of multiple cluster nodes. Based on the two parallel tools, we can concentrate on the parallel design of GFA and ignore some details in parallel implementation, to improve the algorithm efficiency and avoid some unnecessary work.

The MATLAB parallel computing platform is composed of cluster nodes and networks from the view of physical distribution. And it is composed of workers, scheduler and Matlab client in view of the logic. Fig. 2 shows a typical architecture of parallel computing in MATLAB. The main function of scheduler (job manager) aims to manage Matlab workers and compute resources for Matlab job. Workers are the basic logical units to perform parallel computing tasks in MATLAB, which correspond to physical units, responding to the assigned tasks and returning results [24]. This parallel architecture can be not only deployed on one computer, but also deployed on multiple computers by network.

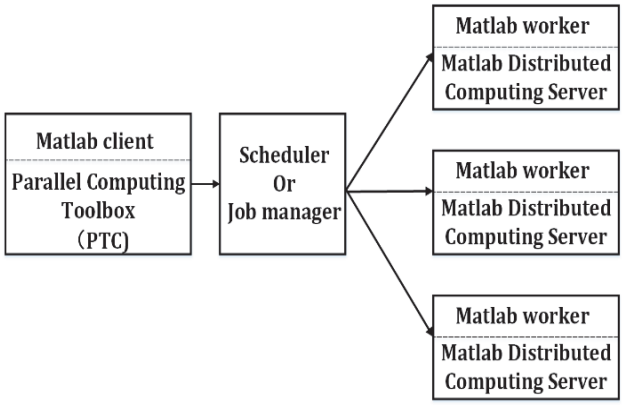

Figure 2 Typical Architecture of Parallel Computing in MATLAB

In this paper, we use the parallel mechanism in MATLAB to realize PGFA in view of the following points:
(1) The platform is established easily. Parallel tools can be installed in one-click without complex environment configuration.
(2) The parallel mode is flexible and diverse. MATLAB provides us with a variety of built-in parallel structures. These parallel structures are provided through system built-in or system functions [24].
(3) The resources of hardware environment are accessible. It is convenient and accessible to use parallel tools with limited hardware resources.

Compared with other parallel platforms, such as CUDA [25] and FPGA [26], parallel platform used in this paper may not have the best performance, but it is quick and easy to design and implement PGFA.

3.3 PGFA Design Based on Parallel Architecture in MATLAB

Based on the analysis of GFA and parallelism architecture in MATLAB, this paper adopts the master-slave model to design PGFA.

Although GFA decomposes solution space, the iterative calculation of each dust particle is still executed serially. Therefore, we divide the initial dust population into several subgroups, and these subgroups are performed in parallel respectively, to improve the efficiency of GFA. The flow chart of PGFA is shown in Fig. 3.

The main steps of PGFA are summarized as follows:

- Initiating dust population. Dust particles are initialized randomly, which is the same as the original GFA.
- Being grouped randomly. This is indistinct from the process in original GFA. In PGFA, subgroups are stored separately and fed into different executors for subsequent iteration.
- Parallel computation in subgroups. The parallel computation in subgroups consists of dust moving, absorbing and rotating, and then returns an optimal solution of current subgroup. When the current subgroup converges, the optimal solution is recorded. When the computation in a subgroup is completed, dust particles in current group will jump out of the intra-group loop without waiting for dust particles in other workers.
- Summarizing the results. The dust particles of each subgroup will be integrated. The optimal results of all subgroups are also collected, and the optimal solution of the whole algorithm is picked among those results.
- Grouping all dust particles again, a new round of iterations will be performed to avoid the algorithm falling into local optimum and ensure the accuracy of the solution. When the termination condition is satisfied, the algorithm ends.

According to the detailed description above, the pseudo-code of PGFA is given in Algorithm 2.

Algorithm 2 PGFA

```
1: dusts[] <---- initialization randomly
2: Define a sliced variable: centers
3: groupdusts[] <---- grouping randomly
4: # Starting the parallel computation
5: for i = 1 to the number of subgroups do
6:     getCenter();  // To get the center dust
7:     moveing();   // Moving dust
8:     absorbing();
9:     rotating();    // Rotation operation
10: end for
11: # Ending the parallel process
12: Summary the results of all subgroups
13: BestDust = getBest(); // Calculate the optimal solution
14: if Finish then
15:       return the best dust;
16: else
17:     goto [3] to grouping randomly again;
18: end if
```

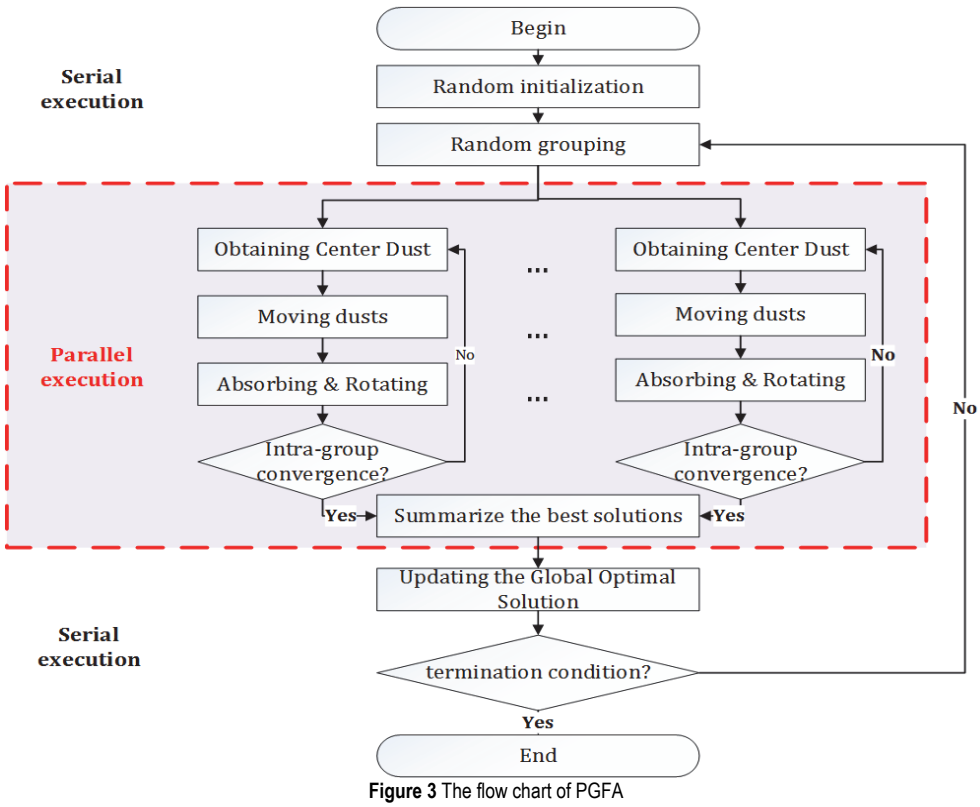

Figure 3 The flow chart of PGFA

3.4 Implementation of PGFA Based on the Parallel Architecture of MATLAB

The parallelism of GFA is essentially due to the independence of individual behavior in GFA. In this paper, we design and implement the PGFA based on the parallel topology of MATLAB with single cluster nodes.

In the topology structure of single cluster nodes, we start the parallel computing pool by "matlabpool", and use MATLAB client-worker mode to complete the PGFA. PGFA code starts up at the client and tasks are distributed to workers in parallel.

3.4.1 Divide Into Subgroups

According to the flow chart for PGFA, dust population should be initialized at client firstly. Then dust particles are grouped and distributed to multiple workers for parallel computation. In general, the number of workers is consistent with the number of CPU cores.

The number of subgroups is mainly depended on the number of workers opened by parallel computing pool. As shown in Fig. 4. the number of subgroups should not be greater than the number of workers in parallel computing pool. Without considering data communication, the maximum degree of parallelism will be achieved when the number of dust groups is consistent with the number of workers. Yet, a large amount of data communication will affect the efficiency of workers in this parallel structure.

During the parallel process, dust particles in each sub-group are updated independently, including the operation of movement, rotation and absorption, and then the optimal solution in subgroup is recorded. At the end of parallel computation of multiple subgroups, the optimal solution of each group is aggregated and the global optimal is selected by comparing with others. At the same time, all dust particles in each subgroup are aggregated too. Finally, the algorithm judges whether the number of iterations has been reached. If not, a new round of algorithm iteration will be carried out, and the newly acquired total dust population will be regrouped as the Fig. 3 shows, to avoid the local optimum.

3.4.2 The Process of Parallel Computation

To ensure the efficiency of PGFA, the process can be designed and implemented in parallel when two conditions are satisfied. One is the computations of one process are intensive. The other one is the loose coupling of data interaction. The computation in subgroups should be independent to reduce the time of communication. So, the initial dust data is set as sliced variable. In this way, it not only meets the requirements of parallel structure for variables, but also saves data transmission time.

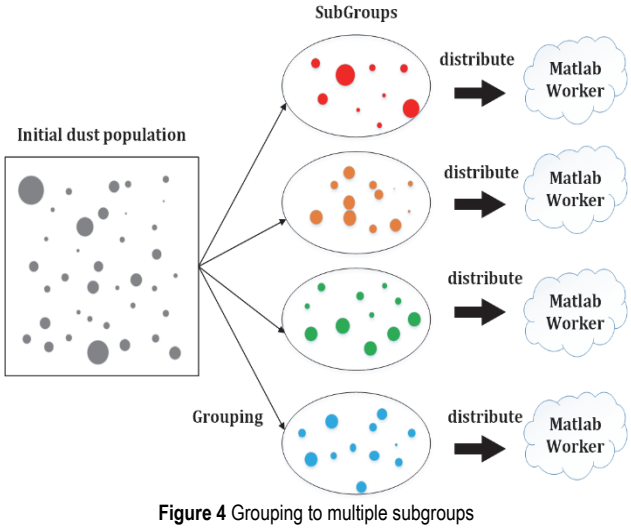

Figure 4 Grouping to multiple subgroups

This shared sliced variable needs to be initialized outside the parallel part, because the variables inside the parallel part are temporary variables, whose values cannot be stored once the parallel process has been finished. The shared sliced variable stores the temporary optimal solution obtained by subgroups for each iteration. Then, the global optimal solution of the algorithm can be obtained by comparing all the optimal solutions stored in the shared sliced variable. The sliced variable as a medium pays an important role in data transmission between serial and parallel programs.

3.4.3 Improvement of Absorption Operation

In addition to improving the efficiency of GFA execution by means of multi-population parallel computation, this paper also makes some improvements on absorption operation.

In the original GFA, center dust will absorb the surrounding dusts which are particularly close to it. The absorption strategy of original algorithm is to delete surrounding dust directly. In the moving operation of original GFA, all the dusts move towards their center dust or generate a new center dust near the original center dust. But with the dust population converging to optimal solution, the surrounding dust particles which are far away from center dust are difficult to become the new center dust. Therefore, the surrounding dust particles which are too far away from center dust will not be a new center dust directly. In this paper, the improved absorption operation not only absorbs the dust particles close to center dust, but also absorbs the remote surrounding dust. There is a threshold as hyper-parameter to control the absorption area. The remote surrounding dust particles exceeding the threshold will be deleted directly. The process of improved absorption operation is illustrated in Fig. 5.

Additionally, the threshold cannot be too small. If the center dust is far away from the theoretical optimal solution and the threshold is small, the dust particles close to theoretical optimal solution will be absorbed and a pseudo-optimal solution will be obtained.

4 EXPERIMENTS AND RESULTS

This part will introduce the details of experiments, including description of experimental platforms, setting of parameters, benchmark functions and experimental results. Finally, this part will give a simple analysis of experimental results.

4.1 Experimental Platforms

PGFA is implemented mainly based on the parallel toolbox in MATLAB. All the experiments were carried out on a computer with 4-core CPU and 8 GB memory. The computing platform in this paper for all the experiments is shown in Tab. 1.

Table 1 Experimental platform

Name	Model
CPU	Intel(R)Core(TM)i5-7200U CPU
Memory(RAM)	8.00GB
Operating System	Windows10,64 bit
Experimental Software	MATLAB 2017b

4.2 Benchmark Function

To test the performance of PGFA in this paper, we select the following eight classical benchmark functions and compare the accuracy of PGFA with the original GFA.

Table 2 Eight benchmark functions

Name	Objective function	Optima
Sphere	$f(x)=\sum_{i=1}^{d} x_i^2$	$x^*=(0,\ldots,0)$ $f(x^*)=0$
Griewangk	$f(x)=\sum_{i=1}^{d}\frac{x_i^2}{4000}-\prod_{i=1}^{d}cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	$x^*=(0,\ldots,0)$ $f(x^*)=0$
Zakharov	$f(x)=\sum_{i=1}^{d} x_i^2+\left(\sum_{i=1}^{d}0.5ix_i\right)^2+\left(\sum_{i=1}^{d}0.5ix_i\right)^4$	$x^*=(0,\ldots,0)$ $f(x^*)=0$
Ackley	$f(x)=20+\mathrm{e}-20\mathrm{e}^{\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\right)}-\mathrm{e}^{\left(\frac{1}{d}\sum_{i=1}^{d}cos(2\pi x_i)\right)}$	$x^*=(0,\ldots,0)$ $f(x^*)=0$
Rotated Hyper-Ellipsoid	$f(x)=\sum_{i=1}^{d}\sum_{j=1}^{i}x_j^2$	$x^*=(0,\ldots,0)$ $f(x^*)=0$
Levy	$f(x)=\sin^2(\pi w_1)+\sum_{i=1}^{d-1}(w_i-1)^2\left[1+10\sin^2(\pi w_i+1)\right]+$ $+(w_d-1)\left[1+\sin^2(2\pi w_d)\right]$ where, $w_i=1+\frac{x_i-1}{4}$	$x^*=(1,\ldots,1)$ $f(x^*)=0$
Rastrigin	$f(x)=10d+\sum_{i=1}^{d}\left[x_i^2-10\cos(2\pi x_i)\right]$	$x^*=(0,\ldots,0)$ $f(x^*)=0$
Dixon-price	$f(x)=(x_1-1)^2+\sum_{i=2}^{d}i\left(2x_i^2-x_{i-1}\right)^2$	$x_i=2^{-\frac{2^i-2}{2^i}}$, for $i=1,\ldots,d$, $f(x^*)=0$

Eight benchmark problems are chosen from a number of significant past studies in unconstraint optimization [27]. The eight test functions are shown in Tab. 2. Where x_i represents the value of x under the dimension i.

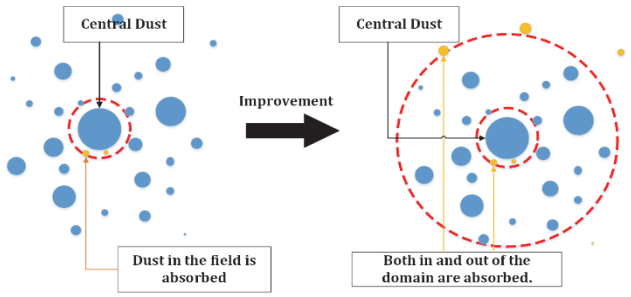

Figure 5 The improved absorption operation

4.3 Parameter Setting

In order to evaluate the experimental results and avoid the influence of environmental randomness on experimental results, 30 independent tests were run for each test function.

Based on the experimental computing platform showed in Tab. 1, four cores are accessible for parallel computing, so the dust particles are divided into 4 subgroups in this paper.

Table 3 The parameter setting for GFA and PGFA

Parameters	GFA PGFA
Dimension	20/50/100
The maximum number of iteration	4 000
The number of groups	4
The population size	5 000

The parameters and their values for the algorithms are given in Tab. 3. In this paper, the dimension of search space is set as 20, 50 and 100 respectively, and three groups of comparative experiments are tested on eight test functions. For each dimension, the initial number of dust particles is set as 5000. In order to compare the algorithm efficiency on different problems, we set the same maximum number of iterations as 4000.

Due to the limitation of computational resources, in order to avoid too long running time, different search spaces are set with different dimensions for PGFA and original GFA. Different test functions have different search spaces in three dimensions as shown in Tab. 4.

Table 4 The search spaces of 8 benchmark functions

Benchmark Function	Range of variables		
	20 dims	50 dims	100 dims
Sphere	[−50, 50]	[−10, 10]	[−10, 10]
Griewangk	[−30, 30]	[−10, 10]	[−10, 10]
Zakharov	[−20, 20]	[-5, 5]	[−5, 5]
Ackley	[−30, 30]	[−30, 30]	[−10, 10]
Rotated Hyper-Ellipsoid	[−50, 50]	[−10, 10]	[−5, 5]
Levy	[−10, 10]	[−10, 10]	[−10, 10]
Rastrigin	[−1, 1]	[−1, 1]	[−1, 1]
Dixon-price	[−10, 10]	[−5, 5]	[−5, 5]

4.4 Results and Discussion

The experimental results in different dimensions are shown in Tab. 5. It shows the average value of optimal solution and the average running time on 30 independent trials obtained by original GFA and PGFA in 20/50/100 dimensions.

In Tab. 5, "Mass" represents the optimal solution obtained by PGFA and original GFA, and "Time" denotes the running time of the algorithms. All results are the average value on 30 independent tests. The comparison of the results indicates that the efficiency of PGFA is generally better than original GFA. But with the increase of dimensions, the accuracy of the two algorithms is decreased.

Table 5 Average results on 30 independent tests

Function	Algorithm	20 dims		50dims		100dims	
		Mass	Time / s	Mass	Time / s	Mass	Time / s
Dixon_price	GFA	6.67E−01	468.807	7.05E−01	320.304	6.11E+01	561.081
	PGFA	6.67E−01	210.335	6.91E−01	154.902	1.96E+01	222.141
Rastrigin	GFA	1.36E+01	665.806	4.27E+01	400.088	1.26E+02	563.763
	PGFA	4.03E−01	272.769	2.81E+00	189.599	1.57E+01	258.869
Ackley	GFA	1.06E+00	602.514	3.58E+00	496.999	2.63E+00	571.156
	PGFA	4.43E−03	237.265	3.40E+00	195.953	2.60E+00	235.592
Griewangk	GFA	1.89E−06	389.414	1.52E−02	333.121	2.01E−01	560.273
	PGFA	1.71E−06	165.387	5.06E−05	143.816	4.53E−02	257.805
Levy	GFA	2.72E−02	621.448	1.43E+00	468.791	7.21E+00	761.625
	PGFA	2.39E−02	258.461	1.14E+00	187.101	5.20E+00	359.546
Rotated Hyper-Ellipsoid	GFA	1.34E+00	393.878	1.22E+01	332.084	4.25E+01	610.462
	PGFA	2.84E−01	101.603	4.88E−01	143.123	1.73E+00	276.174
Sphere	GFA	2.44E−05	518.564	1.52E−02	380.044	1.37E+01	532.579
	PGFA	2.46E−5	210.827	3.30E−04	196.634	2.62E+00	236.864
Zakharov	GFA	1.57E−01	263.861	8.42E−02	399.202	8.71E+00	678.988
	PGFA	1.61E−04	112.607	5.28E−04	175.338	1.12E+00	306.782

In order to show the experimental results better, the following Fig. 6 Fig. 7 are given according to the data from Tab. 5.

In order to normalize the data and improve the effect of data visualization. Fig. 6 takes the $\log_{10}(Mass-0)$ as the error, as all optimal values of the eight benchmark functions from Tab. 2 are equal to zero $(error=\log_{10}(Mass-0))$.The shorter the column is, the smaller error the algorithm has. From Fig. 6, it can be seen that PGFA generally has smaller error. So we can conclude that PGFA has better accuracy than GFA, regardless of the dimensions.

Fig. 7 obviously shows that the running time of the PGFA proposed in this paper is much less than that of the original GFA, whether in 20, 50 or 100 - dimensional search space. Vertical axis indicates second.

By comparing the running time of two algorithms, it can be concluded that the speed-up ratio of PGFA compared with original GFA is between 2 and 3, which shows the efficiency of PGFA under the multi-core environment. We also adopt cores usage one by one in 50

dimensions. The specific experimental data are listed in the Appendix (Tab. 1A). The speed-up ratio of each core in Tab. 6 is the average value of the speed-up ratio obtained by PGFA in 8 test functions. S_p represents the speed-up ratio.

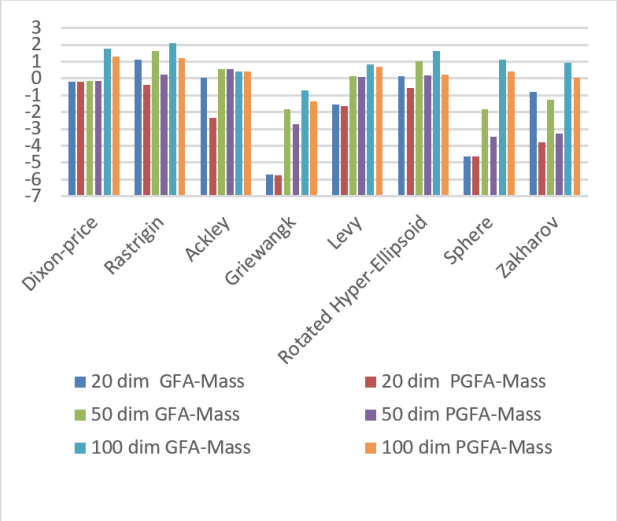

Figure 6 The Average Error on 30 Trials in Different Dimension

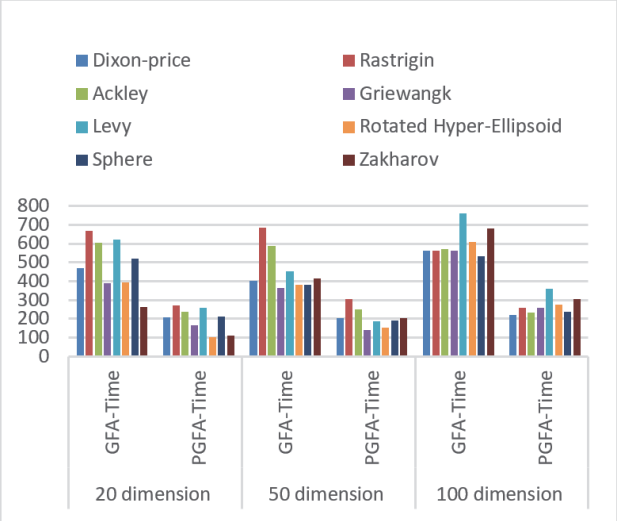

Figure 7 The Average Running Time on 30 Trials in Different Dimension

Table 6 The average speed-up ratio under different number of cores

	1 core	2 cores	3 cores	4 cores
S_p	1.14974	1.87021	1.92600	2.25824

It can be seen from the table results that as the number of cores increases, the acceleration effect is better. However, the best speed-up ratio is lower than the number of cores (4 in this case). Failing to reach the ideal speedup ratio, because PGFA runs the programs serially at the stages of initialization and result summary.

Moreover, each core needs to exchange information with the client, which costs extra time. Therefore, it is difficult to get an ideal acceleration ratio that is equal to the number of cores. At the same time, this problem also suggests that the proposed PGFA still has potential improvement space: on the one hand, we can increase the proportion of parallel codes, on the other hand, we can reduce the data transfer between client and worker as much as possible. In the future work, we can implement GFA in a distributed way of multi-node cluster to achieve better efficiency.

In fact, this paper also tests the optimization results under different dusts numbers (N = 2500, N = 5000, N = 7500). The data is listed in the Appendix (Tab. 2A, Tab. 3A). It shows that PGFA always has less running time than GFA. As the number of dusts increases, both algorithms improve the accuracy of solution, and PGFA has better performance. Once again, it is proved that PGFA has better efficiency than GFA.

5 CONCLUSION

In view of the low efficiency of GFA in serial mode and the long-time of solving large-scale problems, this paper proposes a multi-population parallel GFA in multi-core environment based on the parallel architecture in MATLAB. Multiple subgroups correspond to the core one by one. Meanwhile, the absorption strategy has been improved, based on the original GFA. To justify the performance of proposed PGFA, eight test functions in different dimensions are tested. The experimental results show that PGFA can make full use of the computing power of universal multi-core CPU and improve the efficiency compared with original GFA. At the same time, the parallel strategy and architecture proposed in this paper have good universality and can be adopted to improve the efficiency of similar optimization algorithms.

Acknowledgements

This research was funded by the National Natural Science Foundation of China (Nos. 61772227, 61702214), the Development Project of Jilin Province of China (Nos. 20180414012GH, 20170101006JC, 2020C003). This work was also supported by Jilin Provincial Key Laboratory of Big Data Intelligent Computing (No. 20180622002JC).

6 REFERENCES

[1] Zhu, Y. B. (2014). Overview of Particle Swarm Optimization. *Applied Mechanics and Materials*, *543-537*, 1597-1600. https://doi.org/10.4028/www.scientific.net/AMM.543-547.1597

[2] Wang, Y, Feng X. Y., Huang, Y. X., et al. (2007). A novel quantum swarm evolutionary algorithm and its applications. *Neurocomputing*, *70*(4-6), 633-640. https://doi.org/10.1016/j.neucom.2006.10.001

[3] Fu, M. Y., Askin, R., Fowler, J., et al. (2011). Batch Production Scheduling for Semiconductor Back-End Operations. *IEEE T Semiconduct M*, *24*(2), 249-260. https://doi.org/10.1109/TSM.2011.2114900

[4] Littlejohn, K. A., Hooley, P., & Cox, P. W. (2012). Bioinformatics predicts diverse Aspergillus hydrophobins with novel properties. *Food Hydrocolloid*, *27*(2), 503-516. https://doi.org/10.1016/j.foodhyd.2011.08.018

[5] Koza, J. R. (1992). Genetic programming: on the programming of computers by means of natural selection.

[6] Boussaïd, I., Lepagnot, J. ,& Siarry, P. (2013). A survey on optimization metaheuristics. *Information Sciences*, *237*, 82-117. https://doi.org/10.1016/j.ins.2013.02.041

[7] Dokeroglu, T., Sevinc, E., Kucukyilmaz, T., & Cosar, A. (2019). A survey on new generation metaheuristic algorithms. *Computers & Industrial Engineering*, *137*, 106040. https://doi.org/10.1016/j.cie.2019.106040

[8] Holland, J. H. (1992). Adaptation in Natural and Artificial System. *MIT press*. https://doi.org/10.7551/mitpress/1090.001.0001
[9] Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, man, and cybernetics, Part B: Cybernetics, 26*(1), 29-41. https://doi.org/10.1109/3477.484436
[10] Kennedy, J. & Eberhart, R. (2002). *Particle swarm optimization.* Paper presented at the Icnn95-international Conference on Neural Networks.
[11] Mishchenko, A. S., Matsui, H., & Hasegawa, T. (2012). Distribution of localized states from fine analysis of electron spin resonance spectra of organic semiconductors: Physical meaning and methodology. *Physical Review B Condensed Matter*, *85*(8), 3711-3711. https://doi.org/10.1103/PhysRevB.85.085211
[12] Kaliuzhnyi-Verbovetskyi, D. S. (2012). Noncommutative rational functions, their difference-differential calculus and realizations. *Multidimensional Systems & Signal Processing*, *23*(1-2), 49-77. https://doi.org/10.1007/s11045-010-0122-3
[13] Zheng, M., Liu, G., Zhou, C., Liang, Y., & Wang, Y. (2010). Gravitation field algorithm and its application in gene cluster. *Algorithms for Molecular Biology*, *5*(1), 32-32. https://doi.org/10.1186/1748-7188-5-32
[14] Huang, L., Hu, X., Wang, Y., Zhang, F., Liu, Z., & Pang, W. (2017). *Gravitation field algorithm with optimal detection for unconstrained optimization.* Paper presented at the 2017 4th International Conference on Systems and Informatics (ICSAI), 1411-1416. https://doi.org/10.1109/ICSAI.2017.8248508
[15] Wang, J., Wang, Y., Zhang, C., Du, W., Zhou, C., & Liang, Y. (2009). *Parameter selection of support vector regression based on a novel chaotic immune algorithm.*Paper presented at the 2009 Fourth International Conference on Innovative Computing, Information and Control (ICICIC), 652-655. https://doi.org/10.1109/ICICIC.2009.287
[16] Safronov, V. S. (1972). *Evolution of the Protoplanetary Cloud and Formation of the Earth and the Planets*. Israel Program for Scientific Translations Jerusalem.
[17] Zheng, M., Sun, Y., Liu, G.-x., Zhou, Y., & Zhou, C.-g. (2012). Improved gravitation field algorithm and its application in hierarchical clustering. *PloS one*, *7*(11), e49039. https://doi.org/10.1371/journal.pone.0049039
[18] Choong, A., Beidas, R., & Zhu, J. (2010). *Parallelizing simulated annealing-based placement using GPGPU.* Paper presented at the 2010 International Conference on Field Programmable Logic and Applications, 31-34. https://doi.org/10.1109/FPL.2010.17
[19] Waintraub, M., Schirru, R., & Pereira, C. M. (2009). Multiprocessor modeling of parallel Particle Swarm Optimization applied to nuclear engineering problems. *Progress in Nuclear Energy*, *51*(6-7), 680-688. https://doi.org/10.1016/j.pnucene.2009.02.004
[20] Ziegel, E. (2002). Genetic Algorithms and Engineering Optimization. *Technometrics*, *44*(1), 95-95. https://doi.org/10.1198/tech.2002.s675
[21] Cai, Y., Guang-Yao, L. I., & Wang, H. (2013). Research and implementation of parallel particle swarm optimization based on CUDA. *Application Research of Computers*, *30*(8), 2415-2418
[22] Huang, F. & Fan, X. P. (2006). Parallel Particle Swarm Optimization Algorithm with Island Population Model. *Control & Decision*, *21*(2), 175-166.
[23] Yuan, S., Huang, X. C., & Yang, X. (2010). Configuration and Application of Matlab Parallel Computing Cluster Under Windows Environment. *Computer & Modernization*, *5*,051.
[24] Luszczek, P. (2009). Parallel Programming in MATLAB. *The International Journal of High Performance Computing Applications*, *23*(3), 277-283. https://doi.org/10.1177/1094342009106194
[25] Garland, M., Grand, S. L., Nickolls, J., Anderson, J., Hardwick, J., Morton, S., & Volkov, V. (2008). Parallel Computing Experiences with CUDA. *Micro IEEE*, *28*(4), 13-27. https://doi.org/10.1109/MM.2008.57
[26] Li, D., Huang, L., Wang, K., Pang, W., Zhou, Y., & Zhang, R. (2018). A General Framework for Accelerating Swarm Intelligence Algorithms on FPGAs, GPUs and Multi-Core CPUs. *IEEE Access*, *6*,72327-72344. https://doi.org/10.1109/ACCESS.2018.2882455
[27] Zames, G., Ajlouni, N. M., Ajlouni, N. M., Ajlouni, N. M., Holland, J. H., Hills, W. D., & Goldberg, D. E. (1981). Genetic algorithms in search, optimization and machine learning. *Information Technology Journal*, *3*(1), 301-302.

Contact information:

Lan HUANG, professor
Jilin University,
College of Computer Science and Technology,
Changchun, CN 130012
E-mail: huanglan@jlu.edu.cn

Wen-Xuan WU, master
Jilin University,
College of Computer Science and Technology,
Changchun, CN 130012
E-mail: 2715991564@qq.com

Xue-Mei HU, doctoral student
(Corresponding author)
Jilin University,
College of Computer Science and Technology,
Changchun, CN 130012
E-mail: 1432466560@qq.com

Sen YANG, doctoral student
Jilin University,
College of Computer Science and Technology,
Changchun, CN 130012
E-mail: yangsen18@mails.jlu.edu.cn

Yu LIN, doctoral student
Jilin University,
College of Artificial Intelligence,
Changchun, CN 130012
E-mail: linyu19@mails.jlu.edu.cn

Yan WANG, professor
(Corresponding author)
Jilin University,
College of Computer Science and Technology,
Changchun, CN 130012
E-mail: wy6868@jlu.edu.cn

Appendix

Table 1A 50 dimensional algorithm running time

Function	GFA-Time	PGFA-Time			
		1 core	2 cores	3 cores	4 cores
Dixon-price	320.304	373.363	195.047	190.999	154.902
Rastrigin	400.089	417.366	260.792	234.273	189.599
Ackley	496.999	287.098	227.105	200.415	195.953
Griewangk	333.121	300.571	177.096	173.529	143.816
Levy	468.791	381.367	236.083	222.508	187.101
Rotated Hyper-Ellipsoid	332.084	302.584	186.267	184.306	143.123
Sphere	380.044	335.185	220.025	199.611	196.634
Zakharov	399.202	369.124	228.847	203.074	175.338

Table 2A Algorithm running time of different scale in 50 dimensions

Function	$N = 2500$		$N = 5000$		$N = 7500$	
	GFA-Time	PGFA-Time	GFA-Time	PGFA-Time	GFA-Time	PGFA-Time
Dixon-price	96.639	37.907	361.784	193.058	708.891	375.945
Rastrigin	101.704	42.088	454.363	231.826	835.875	468.581
Ackley	99.459	39.580	406.307	148.475	809.155	379.398
Griewangk	102.681	37.734	333.967	140.319	686.838	342.915
Levy	127.348	55.540	439.943	192.305	924.486	426.462
Rotated Hyper-Ellipsoid	106.957	41.413	358.48	157.974	741.049	368.21
Sphere	95.972	68.251	315.232	182.149	690.133	383.031
Zakharov	112.758	52.515	371.573	211.632	744.359	418.012

Table 3A Algorithm optimization results of different scalein 50 dimensions

Function	$N = 2500$		$N = 5000$		$N = 7500$	
	GFA-Mass	PGFA-Mass	GFA-Mass	PGFA-Mass	GFA-Mass	PGFA-Mass
Dixon-price	4.36E+00	8.79E+00	6.79E−01	6.76E−01	6.76E−01	6.75E−01
Rastrigin	2.23E+00	2.49E+01	4.27E+01	2.05E+00	1.05E+00	5.06E−02
Ackley	4.59E+00	3.84E+00	4.17E+00	2.55E+00	2.68E+00	3.61E+00
Griewangk	1.60E−01	3.00E−02	3.40E−03	2.29E−04	1.19E−03	1.05E−05
Levy	1.82E+00	1.62E+00	7.70E−01	1.08E+00	8.15E−01	6.84E−01
Rotated Hyper-Ellipsoid	7.39E+01	1.59E+01	2.17E+01	8.06E−02	2.38E−02	1.23E−02
Sphere	1.80E+00	2.58E−01	7.19E−03	2.72E−04	2.99E−04	2.54E−04
Zakharov	3.09E+00	6.88E−01	9.65E−02	4.90E−04	6.15E−04	4.65E−04