Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 100 (2015) 1672 – 1677

Procedia Engineering

www.elsevier.com/locate/procedia

25th DAAAM International Symposium on Intelligent Manufacturing and Automation, DAAAM 2014

A Time Performance Evaluation of the Soma Asynchronous Parallel Distribution in Java and C#

Jan Kolek*, Roman Jasek

Tomas Bata University in Zlin, nam. T.G.Masaryka 5555, 760 01 Zlin, Czech Republic

Abstract

This paper compares two different implementations of the Self-Organizing Migrating Algorithm (SOMA), which is a highly effective tool of an evolutionary optimization that is aimed at the same set of problems as Genetic Algorithms. One implementation of algorithm was created in the C# framework and the second implementation in Java framework. Both implementations are the asynchronous parallel 'All-to-One' strategy of SOMA, which is used to equally distribute computation loads between several available processors/cores. The aim of our effort is to statistically evaluate the computation time efficiency of these two concurrent frameworks including dependence on the number of threads. The obtained results are discussed in the conclusion.

Keywords: asynchronous; evolutionary algorithm; parallel; optimization; SOMA

1. Introduction

This paper belongs into area of optimization by artificial intelligence. One of many optimization methods is Self-Organizing Migrating Algorithm alias SOMA. Our primary aim is compares two different implementations of the SOMA. One implementation of algorithm was created in the C# framework and the second implementation in Java framework. In the first test the dependence of solution quality on the number of used threads was evaluated and in the second test the consumption of time was evaluated.

The SOMA is a very effective tool of evolutionary optimization. It was created in 1999. Algorithm is ranked among evolution algorithms although there are not created new individuals during running the algorithm, in

* Corresponding author. Tel.: +420 57 603 5274.
E-mail address: kolek@fai.utb.cz

doi:10.1016/j.proeng.2015.01.542

contradiction with typical evolution algorithms. [11] Only the coordinates of individuals are changed in the area of possible solution. Therefore, SOMA can be classified as memetic algorithm or among swarm algorithms more accurately. The SOMA is inspired by the intelligent behaviour of groups of individuals in the nature, e. g. when they searching food or finding the shortest way towards it. [8, 9, 10]

As it was already mentioned, the new individuals are not created by selective breeding but the algorithm only changes coordinates of individuals. The evolution cycle, which is called „generation" at other Genetic Algorithms, was renamed to „migration loop" in the SOMA. [3, 5] There are several strategies of elementary settings of SOMA. One is the asynchronous parallel All-To-One, where all individuals are moved to the main individual which is called Specimen. This strategy is used in our tests. [2, 7, 12]

2. SOMA

2.1. Parameter definition

Before starting the algorithm, SOMA's parameters: Step, PathLength, PopSize, PRT and a Cost Function needs to be defined. The run of the algorithm is influenced by settings of parameters indicated in Table 1. [1, 11]. The Cost Function is simply the function which returns a scalar that can directly serve as a measure of fitness.

Table 1. SOMA parameters.

Name of parameter	Recommended range	Comment
PathLength	[1.1;>5]	Control parameter
Step	[0.11;PathLength]	Control parameter
PTR	[0;1]	Control parameter
D	Dimension	Dimension of the problem
PopSize	[10;define the user]	Control parameter
Migrations	[10;define the user]	Termination parameter
MinDiv	[±arbitrary;define the user]	Termination parameter

2.2. Creation of the population

The population of individuals is randomly generated. Each parameter for every individual has to be randomly chosen from a given range <Low, High>.

2.3. Migration loop

All individuals from population (PopSize) are evaluated by the Cost Function and the Leader (individual with the highest fitness) is chosen for a current migration loop. Subsequently, remaining individuals begin to jump, (according to the Step definition) towards the Leader. Each individual is evaluated after every jump using the Cost Function. Jumping continues until a final position defined by the PathLength is reached. New position xi,j after each jump is calculated by (1). This is shown graphically in Fig. 1. Individual returns then to that position where they found the best fitness on their trajectories. [1, 7]

$$x_{i,j}^{MLnew} = x_{i,j,start}^{ML} + (x_{L,j}^{ML} - x_{i,j,start}^{ML})tPRTVector_j$$
$$where\ t \in <0, by\ Step\ to, PathLength>$$
$$and\ ML\ is\ an\ actual\ migration\ loop$$

Before an individual begins its jumping towards the Leader, a random number rnd is generated (for each individual's coordinate), and then compared with PRT. If a random number generated is larger than PRT, an associated coordinate of the individual is set to 0 by the means of PRTVector.

$$\begin{gathered} if\ rnd_j < PRT\ then\ PRTVector_j = 0, \\ else\ 1 \\ where\ rnd\ \in < 0,1 > and\ j = 1,2, \dots n \end{gathered}$$

Hence, the individual moves in the N-k dimensional subspace, which is perpendicular to the original space. This fact establishes a higher robustness of the algorithm. Earlier experiments have demonstrated that, without the use of PRT, SOMA tends to determine a local optimum rather than the global one. [2, 4]

2.4. Test for stopping condition

If a maximum number of migration loops has been reached, stop and recall the best solution(s) found during the search.

3. Experimental settings for evaluation of performance

The aim of the experiment is a performance evaluation of asynchronous parallel algorithm SOMA based on classical one-thread strategy All-To-One. The SOMA which was tested was modified so that it not used only one thread but it used just as many threads how many processor cores is available. The proposal and description of such implementation of SOMA was firstly published and can be found in [6].

The asynchronous SOMA (in contrast to synchronous version of the algorithm) does not wait for all individuals to finish their paths to Leader. If any individual gets better position than the current Leader, this individual becomes promptly the new Leader. The other individuals continue their path towards this new Leader promptly, not towards the original Leader. This method significantly accelerates and increases the opportunity to finding global extreme value. [1,4]

The experiment was performed on the personal computer from the ACER company with two-core processor AMD Athlon 4960 (64-bit) and the operating memory 3 GB DDR2 under operating system Windows Vista 64-bit. The asynchronous parallel SOMA was tested for these 10 test functions, proposed as the benchmark. The functions are shown there for better clarity:

$$\sum_{i=1}^{Dim-1}(20 + E - 20E^{-0,2\sqrt{0.5(x_i^2+x_{i+1}^2)}} - E^{-0.5(\cos(2\pi x_i)+\cos(2\pi x_{i+1}))}) \tag{1}$$

$$\sum_{i=1}^{Dim-1}(-x_i sin(\sqrt{|x_i-(x_{i+1}+47)|}) - (x_{i+1}+47)sin(\sqrt{\left|x_{i+1}+47+\frac{x_i}{2}\right|})) \tag{2}$$

$$1 + \sum_{i=1}^{Dim}\frac{x_i^2}{4000} - \prod_{i=1}^{Dim} cos(\frac{x_i}{\sqrt{i}}) \tag{3}$$

$$-\sum_{i=1}^{Dim-1}(E^{\frac{-(x_i^2+x_{i+1}^2+0.5x_i x_{i+1})}{8}} cos(4\sqrt{x_i^2+x_{i+1}^2+0.5x_i x_{i+1}})) \tag{4}$$

$$\sum_{i=1}^{Dim-1}(-1(sin(x_i)sin\left(\frac{x_i^2}{\pi}\right)^{20} + sin(x_{i+1})sin(\frac{2x_{i+1}^2}{\pi})^{20})) \tag{5}$$

$$\sum_{i=1}^{Dim-1}\left(x_i sin(a)cos(b) + (x_{i+1}+1)sin(b)cos(a)\right) \\ where\ a = \sqrt{|x_{i+1}+1-x_i|}, b = \sqrt{|x_{i+1}+1+x_i|} \tag{6}$$

$$(Dim * 10)\sum_{i=1}^{Dim}(x_i^2 - 10cos(2\pi)) \tag{7}$$

$$\sum_{i=1}^{Dim-1}(100(x_i^2 - x_{i+1}^2)^2 + (1-x_i^2)^2) \tag{8}$$

$$\sum_{i=1}^{Dim-1} -x_i sin\left(\sqrt{|x_i|}\right) \tag{9}$$

$$-\sum_{i=1}^{Dim-1}(x_i^2 + x_{i+1}^2)^{0.25} sin((50(x_i^2 + x_{i+1}^2)^{0.1})^2 + 1) \tag{10}$$

All tests were performed in 100 dimensional space (each function has 100 coordinates) and the process of optimization was 100-times repeated. In all cases, new initial population was again generated as starting point of the optimization. The parameters for tests were set on Step = 0.11 (Rosenbrock Step= 0.011) and PRT = 0.1.

Table 2. Parameters of SOMA used in the experiment.

Test function		PathLength	PopSize	Min	Max	Real extreme (minimum)
Ackley	(1)	3	100	-20	20	0
EggHolder	(2)	3	60	-512	512	not known
Griewangk	(3)	3	100	-50	50	0
Masters	(4)	3	60	-5	5	-100
Michalewicz	(5)	0,5	60	0	3	98,10
Rana	(6)	3	100	-512	512	not known
Rastrigin	(7)	3	100	-5	5	-20000
Rosenbrock	(8)	0,5	60	-3	3	0
Schwefel	(9)	3	60	-512	512	-41898,3
Sine Wave	(10)	0,5	100	-10	10	0

Table 2. describes others parameters which were used in tests and also expected optimal results (real extreme) of used benchmark functions (1) – (10).

4. Evaluation the dependence of solution quality on the number of used threads

The founding solutions for both implementations of the SOMA (C# and Java) are shown in Table 3. It provides a comparison of quality solutions from both implementations and the better solution is in bold.

Table 3. Comparison implementations of algorithm.

Test function		Java 1thread	Java 2threads	C# 1thread	C# 2threads	Real extreme (minimum)
Ackley	(1)	**236.82**	244.31	245.92	237.51	0
EggHolder	(2)	-54872.35	-46925.03	-56327.15	**-62374.70**	not known
Griewangk	(3)	**0.01**	0.09	0.08	0.03	0
Masters	(4)	-91.31	-91.17	-88.47	**-91.62**	-100
Michalewicz	(5)	-87.53	**-90.84**	-85.05	-88.49	-98,10
Rana	(6)	-23679.98	-27413.65	-22513.29	**-32424.71**	not known
Rastrigin	(7)	-187802.35	-194313.64	**-947986.86**	-173362.87	-200000
Rosenbrock	(8)	854.41	**497.49**	731.14	551.27	0
Schwefel	(9)	-39871.42	**-41540.04**	-40297.83	-41068.55	-41898,3
Sine Wave	(10)	57.19	59.28	63.46	**55.83**	0

As be seen in Table 3 implementation in Java was best in 2 causes for one thread and in 3 causes for two threads. Implementation in C# was best in 1 cause for one thread and in 4 causes for two threads. It can mean that there is not dependence finding of best solution on number of used threads and used implementation. Finding of good solution depends on the random start population of optimization rather than on used implementation of SOMA. Both implementations are equal and so we can compare time consumption of both implementations.

5. Evaluation the time consumption

The main aim of this paper was to compare which of two implementations (Java, C#) has better computational time. The results are shown in a table 4. The table shows that the asynchronous parallel SOMA which was created in C# is better in 7 cases from 10. For example, in the case Michalewicz test function, which is most calculation demanding from the used benchmark function, it was more than two times faster. By way of contrast, the asynchronous parallel SOMA created in Java was marginally faster only in 3 cases.

Table 4. Comparison of time consumption.

Test function		Java 1thread time [s]	Java 2 threads time [s]	C# 1 thread time [s]	C# 2 threads time [s]
Ackley	(1)	77.89	40.20	43.21	**21.81**
EggHolder	(2)	21.73	11.12	12.59	**6.69**
Griewangk	(3)	10.73	**5.93**	17.45	9.07
Masters	(4)	45.37	23.32	33.30	**16.67**
Michalewicz	(5)	92.62	46.80	42.81	**21.61**
Rana	(6)	37.98	19.36	20.60	**10.29**
Rastrigin	(7)	11.91	**6.74**	15.27	7.74
Rosenbrock	(8)	12.02	**11.16**	28,25	14.32
Schwefel	(9)	11.20	6.15	9,01	**4.60**
Sine Wave	(10)	84.96	43.57	61,93	**31.15**

Conclusion

Table 3. shows results for evaluate the dependence of solution quality on the number of used threads. The experiment verified that both implementations of asynchronous parallel SOMA are equal from the point of view a quality of found solution. Both implementations of SOMA won in 5 cases from 10. Finding of good solution depends on the random start population of optimization rather than on used implementation of SOMA.

Regarding the main experiment's result, it can be concluded that the implementation of SOMA created in C# is significantly better in time consumption. How can be seen in Table 4, this implementation was faster in 7 cases from 10 cases for two threads, certainly. For the Michalewicz test function, which is most mathematically complex, it was even faster more than two times. Such results may indicate that C# (.NET Framework) has got a more effective library for calculating complex mathematical functions. Calculating of mathematical functions was the most time-consuming operation for both implementations of SOMA.

Subsequently, we are going to suggest an experiment to prove this hypothesis.

Acknowledgements

This paper is supported by the Internal Grant Agency at TBU in Zlin, projects No. IGA/FAI/2014/027.

References

[1] Zelinka,I.,, "SOMA - Self-Organizing Migrating Algorithm". In New Optimization Techniques in Engineering. Springer, 2004. ISBN: 978-3-540-20167-0.

[2] Senkerik R., Oplatkova Z., Zelinka I., Davendra D. Synthesis of feedback controller for three selected chaotic systems by means of evolutionary techniques: Analytic programming, Mathematical and Computer Modelling, Available online 27 May 2011, ISSN 0895-7177, 10.1016/j.mcm.2011.05.030.

[3] Chramcov B., Beran P., Daníček L., Jašek R.. A simulation approach to achieving more efficient production systems. International Journal of Mathematics and Computers in Simulations, 2011, year 5, issue 4, page 299-309. ISSN 1998-0159.

[4] Král E., Vašek, L., Dolinay V., Čápek P. Usage of Peak Functions in Heat Load Modeling of District Heating System. In Recent Researches in Automatic Control. Montreux : WSEAS Press, 2011, p. 404-406. ISBN 978-1-61804-004-6.

[5] Pospíšilík M., Kouřil L., Motýl I., Adámek M. Single and Double Layer Spiral Planar Inductors Optimisation with the Aid of Self-Organising Migrating Algorithm. In Proceedings of the 11th WSEAS International Conference on Signal Processing, Computational Geometry and Artificial Vision. Venice : WSEAS Press (IT), 2011, p. 272 - 277. ISBN 978-1-61804-027-5

[6] Zelinka I., Studies in Fuzziness and Soft Computing, New York : Springer-Verlag, 2004.

[7] Vařacha P. Innovative Strategy of SOMA Control Parameter Setting. In 12th WSEAS International Conference on Neural Networks, Fuzzy Systems, Evolutionary Computing & Automation. Timisoara : WSEAS press, 2011, p. 70-75. ISBN 978-960-474-292-9

[8] Pivnickova, L., Vasek, V., Dolinay, V., Algorithms in the examination of the postural stability, In procceedings of the 10th WSEAS International Conference on Electronics, Hardware, Wireless and Optical Communications (EHAC'11), Cambridge 2011, pp.374-376, ISBN: 978-1-61804-004-6.

[9] Král E., ET AL, Using PSO Algorithm for Parameter Identification of Simulation Model of Heat Distribution and Consumption in Municipal Heating Network. In Proceedings of the 21st International DAAAM Symposium „Intelligent Manufacturing & Automation: Focus on Interdisciplinary Solutions" Vienna : DAAAM International Vienna , 2010. p. 1043 - 1044. ISBN/ISSN: 978-3-901509-73-5.

[10] Jadlovská, A. – Katalinic, B. – Hrubina, K.: Optimal Control of Complex Processes, In: Proceedings of the 15th International DAAAM Symposium, „Intelligent Manufacturing & Automation: Globalisation-Technology - Men-Nature", 2004, Published by DAAAM International, Vienna, Editor B. Katalinic, pp. 178-179, Austria 2004, ISBN 3-901509-42-9, ISSN 1726-9679.

[11] Donald Davendra, Ivan Zelinka, Magdalena Bialic-Davendra, Roman Senkerik, Roman Jasek, Discrete Self-Organising Migrating Algorithm for flow-shop scheduling with no-wait makespan, Mathematical and Computer Modelling, Volume 57, Issues 1–2, January 2013, Pages 100-110, ISSN 0895-7177, http://dx.doi.org/10.1016/j.mcm.2011.05.029.

[12] Ivan Zelinka, Donald David Davendra, Roman Šenkeřík, Michal Pluháček, Investigation on evolutionary predictive control of chemical reactor, Journal of Applied Logic, Available online 18 November 2014, ISSN 1570-8683, http://dx.doi.org/10.1016/j.jal.2014.11.009.