PARALLEL COMPUTATION PLATFORM FOR SOMA

Miroslav Červenka, Ivan Zelinka
Institute of Process Control and Applied Informatics
Faculty of Technology
Tomas Bata University in Zlín
Mostní 5139 Zlín, Czech Republic
E-mail: {cervenka,zelinka}@ft.utb.cz

KEYWORDS

Parallel computation, cluster, evolutionary algorithm, Self-Organizing Migrating Algorithm, SOMA

ABSTRACT

This paper describes an universal portable platform for parallel computations and a parallelized version of relatively new evolutionary optimization algorithm – the Self-Organizing Migration Algorithm (SOMA). Results of efficiency tests of the platform and also test results of optimization with parallel SOMA are included.

INTRODUCTION

Evolutionary algorithms form a class of stochastic optimization algorithms in which priciples of organic evolution are regarded as rules for optimization. The are often applied to parameter optimization probléme (Bäck, Schwefel, 1993), when specialized techiques are not suitable or standard methods give unsatisfactory results.

Principle of evolutionary algrithms is based on scanning through a space of possible solutions using a population of individuals. Because this could be a very time-demandig process, parallelization might be one of possible solutions. By splitting this task on more computers, the search process can be speeded-up or even improved.

The Self-Organizig Migration Algorithm (SOMA) (Zelinka 2002, 2004) is a kind of relatively new optimization algorithm, whose desing enables its parallel implementation very easily. The aim of this work is to develop a universal platform for parallel applications. This framework runs on ordinary office computers used by students in the IT labs and utilizes their iddle CPU time. In addition, another objective is to use SOMA to develop a general-purpose optimization toolkit.

COMPUTATION PLATFORM

To create a computation cluster, a certain number of computers is needed. In our case 50 standard office computers in configuration Athlon XP 2200+, 512 MiB RAM, interconnected by 100Mbps LAN across the campus network were used.

To manage all this computation power a universal distributed enviroment for parallel applications was developed. This platform was designed to fulfil following requirements: First, there was an idea to utilize the idle CPU time in the IT-labs. This computers are owned by different departments running various operating systems on their machines, therefore Java was chosen to enable simple deployment. Second, it had to be as universal as possible to allow the user to simply create his own application using parallel capabilities without being annoyed with the background problems. Several sample applications and detailed HOW-TOs were written to demonstrate the cluster usage.

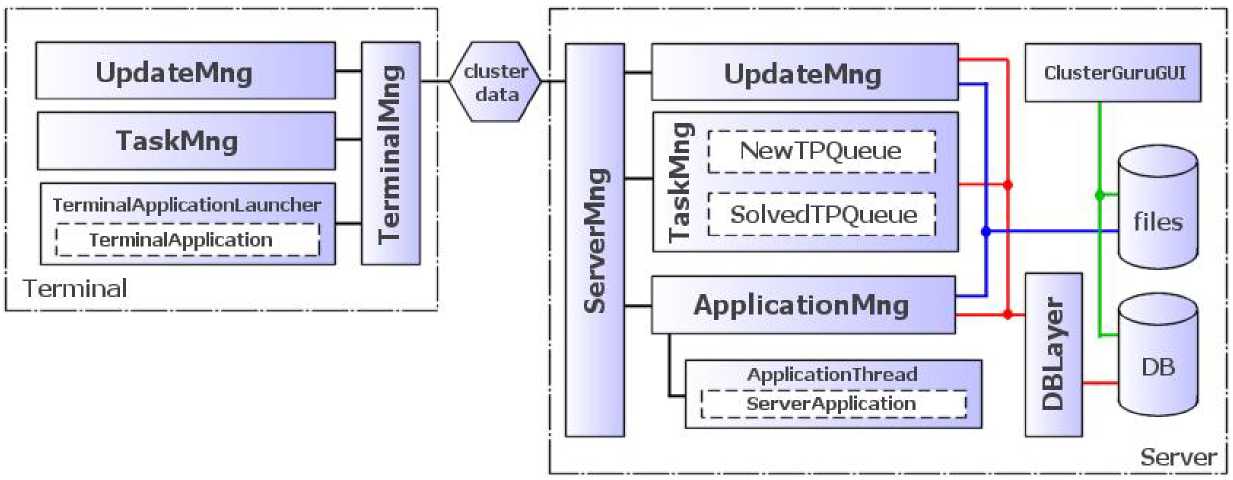


Fig. 1: Cluster structure

CLUSTER STRUCTURE

The cluster platform is splitted into two parts – server and terminal – and the structure can be seen in Figure1.

The server is composed of modules that provide services for the whole cluster. First of them is Update Manager, which ensures that all terminals and also user's classes are up to date on all terminals all the time. The second one is Task Manager, which is responsible for correct task distribution. It handles all task packages created by the user's server application and also all incoming task requests or packages with partial task solutions. It contains two task queues. The flow of task packages between the database and Task Manager is realized in groups of packages and thereby the response time was rapidly reduced. When a terminal fails (for example, it is turned off by a student), its task is assigned to another terminal after a certain interval,which means that it also provides basic fail-over capability. The third module is Server Application Manager, which is responsible for starting and shutting down the user's server application and provides many useful utilities for it, like task distribution or direct connection to terminals. The server is able to handle mutiple instances of different user's server applications.

The terminal is a standalone application which starts just after computer boots. On the Windows platform it runs as an NT system service, so the student using the computer is not disturbed. The terminal was intended as a smart application which can ask server for an update or task to solve. Furthermore, it is able to find other terminals in the network and create a virtual network to eable direct communication with other terminals. As soon as a package with task description is delivered on the terminal, the relevant class able to process this task is found and thereby the user's terminal application is started. While a task is processed, the terminal's task manager requests another package with task description This feature can eliminate time delays caused by communication and might be very useful when transfering huge amount of data. The terminal cannot process concurrent tasks.

PARALLEL APPLICATION

The user's parallel application is also separated into two parts. First of them is the server-side application, which is primarily supposed to manage run of the whole process. It creates packages with description of partial tasks or data needed for the actual computation. Moreover, it decides whether the application run is over or other calculations have to be done. It can directly communicate with one or more terminals and usually does not perform any computations.

The second part is the terminal-side application where the computation operations are performed. When there is not a task to process, the terminal is hibernated and after a longer period of time it performs a request for a task again.

CLUSTER MANAGEMENT

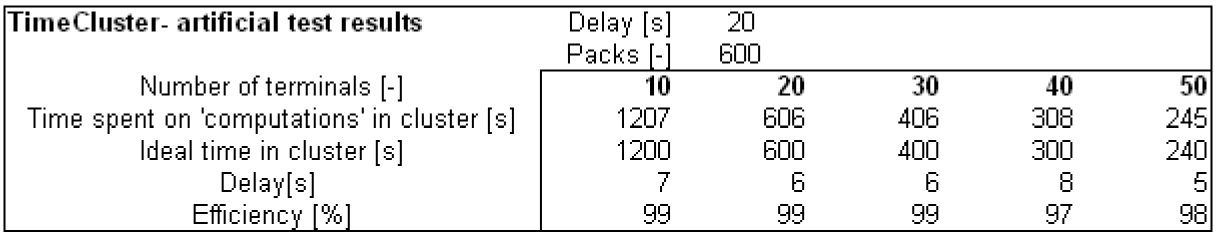
All operations required to manage a user's distributed project are available through a graphical user interface (GUI) presented on the web. This solution was chosen to enable an easy access to the cluster server for anyone using just a web browser and simplify all operations as much as possible. In this interface the parallel application project can be created, modified, launched or terminated. The progress of the application and the time needed to finish it can also be seen here. The results are easily accessible without a need of having a shell access. Another advantage of this solution is that the cluster administration can be accessed from various mobile devices, like cell phones or handhelds.

CLUSTER EFFICIENCY

Quality of distributed enviroment can be considered according to its efectivity. The loss of performance can be caused by server software, where the server modules are not able to handle incoming requests immediately. In some cases the problem could be caused by the user's server application, when it is not able to create task parts fast enough. In addition, the throughput and time delay of the network might be a limitation.

To find the real efficiency of this parallel platform an artificial problem was created. It emulates computation by n-second time delay on terminals. When the number of terminals and also the total computation duration is known, it is very easy co calculate the time needed for execution on certain number of (at least similar) computers. For example – if we have 10 computers and will use 600 task parts with delay of 20 seconds, we can easily calculate the time needed on the number of computers that are available.

Table 1 and Figure 2 display results of the “TimeCluster“ test. For each cluster configuration a virtual problem like in the example described above was created. The average delay was about 6 seconds, which might be due to the task check interval inside the



TimeCluster- artificial test results	Delay [s]	20			
	Packs [-]	600			
Number of terminals [-]	**10**	**20**	**30**	**40**	**50**
Time spent on 'computations' in cluster [s]	1207	606	406	308	245
Ideal time in cluster [s]	1200	600	400	300	240
Delay[s]	7	6	6	8	5
Efficiency [%]	99	99	99	97	98

Tab. 1: TimeCluster test results

platform and could be influenced by settings in the configuration file. This partially decreases the cluster efficiency that can be seen in all columns of Table 1. Even though, the cluster efficiency is high enough to consider the plaform suitable for parallel computations.

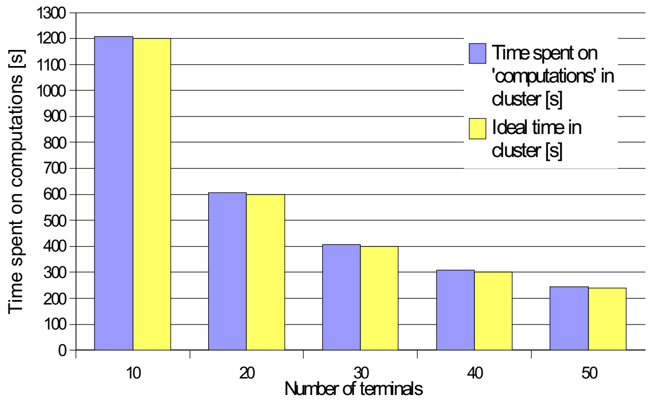


Fig. 2: TimeCluster – artificial test results

PARALLEL SOMA

SOMA immitates competitive-cooperative behaviour of intelligent creatures. A group of wolves or other predators may be a good example. If they are looking for food, they usually cooperate and compete so that if one member of the group is more successful than the previous best one (e.g. has found more food) then all members change their trajectories towards the new most successful member. Exactly this approach is used in the SOMA's One-To-One strategy.

Positions on the hyper-plane are represented by members of the SOMA population – individuals. Each of them stands for an actual solution for the given problem. The best position (the best solution in the particular migration loop) is called leader.

Since evaluation of the optimized cost function for each individual's step can be a time demanding operation, the numerous population of individuals was divided into groups and each group became a separate population on one terminal. Single terminals send a message informing the othes about their local best position – the position of local leader. The information about leader can be either sent from terminals to the server and later back to all terminals, or better, terminals can communicate directly among themselves, which is faster and also more efficient. All features can be set in the relevant configuration files.

While in classic clusters the data is accessible on all nodes at the same time thanks to the shared memory feature, in this case a little different approach is used. The information about the position of the leader spreads over other terminals in steps, as diplayed in Figure 3. After one migration loop the information about the best position is passed only to the neighbouring terminals (to 4 neighbours in this case). If they consider the position to be better as their local leader's position, they will use it as a new local leader and after the next migration loop the information is passed on. The number of neighbours, which influences the information spreading velocity, can be set in the configuration file. This parameter might also have impact on the robustness of the parallelized evolutionary algorithm.

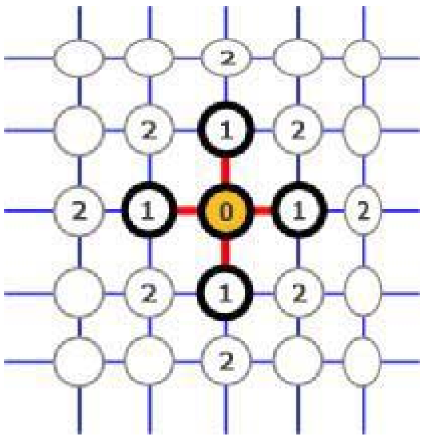


Fig. 3: Spreading of information in the terminal's virtual network

TESTING

Parallelized SOMA was tested on teaching a neural network. Described network configuration and the training set was chosen only for cluster efficiency tests. It has no other meaning. SOMA configuration was: step = 0.11, pathLength = 3, prt = 0.3, minDiv = 0.001, migrations = 4000. Number of modified neuron-wages = 315, population size = dim/2 = 157 individuals. The network had 4 layers with 3 neurons in the input layer, two hidden layers, 15 neurons each, and 3 neurons in the output layer. Hyperbolic cosine was used as transfer function in all neurons. The teaching set had 21 input and 21 output vectors and its structure was as follows:

input: -1, -1, -1; output: -1, -1, -1
input: -0.9, -0.9, -0.9; output: -0.9, -0.9, -0.9
input: -0.8, -0.8, -0.8; output: -0.8, -0.8, -0.8
....
input: 1, 1, 1; output: 1, 1, 1

SOMA – neural network teaching	Single computer time [min]:	230			
Number of terminals [-]	**10**	**20**	**30**	**40**	**50**
Time spent on 'computations' in cluster [min]	23.8	11.8	7.9	5.9	4.7
Ideal time in cluster [min]	23	11.5	7.67	5.75	4.6
Delay [min]	0.8	0.3	0.23	0.15	0.1
Efficiency [%]	97	97	97	97	98

Tab. 2: Neural network taught by SOMA - test results

INPUTS			OUTPUTS		
Input 1	Input2	Input 3	Output 1	Output 2	Output 3
-1.00	-1.00	-1.00	-0.91	-0.92	-0.91
-0.90	-0.90	-0.90	-0.86	-0.87	-0.86
-0.80	-0.80	-0.80	-0.80	-0.80	-0.80
-0.70	-0.70	-0.70	-0.72	-0.72	-0.72
-0.60	-0.60	-0.60	-0.62	-0.61	-0.62
-0.50	-0.50	-0.50	-0.51	-0.50	-0.51
-0.40	-0.40	-0.40	-0.40	-0.39	-0.40
-0.30	-0.30	-0.30	-0.30	-0.30	-0.30
-0.20	-0.20	-0.20	-0.20	-0.21	-0.20
-0.10	-0.10	-0.10	-0.10	-0.11	-0.10
0.00	0.00	0.00	0.00	0.00	0.00
0.10	0.10	0.10	0.10	0.11	0.10
0.20	0.20	0.20	0.20	0.21	0.20
0.30	0.30	0.30	0.30	0.30	0.30
0.40	0.40	0.40	0.40	0.39	0.40
0.50	0.50	0.50	0.51	0.50	0.51
0.60	0.60	0.60	0.62	0.61	0.62
0.70	0.70	0.70	0.72	0.72	0.72
0.80	0.80	0.80	0.80	0.80	0.80
0.90	0.90	0.90	0.86	0.87	0.86
1.00	1.00	1.00	0.91	0.92	0.92

Tab. 3: Output of teaching process

The results performed by the parallel version can be seen in Table 2, progress of global error history in Figure 4. The quality of teaching process (in Table 3) can be considered as very high. Only values around -1 and 1 are not accurate, which is caused by the behaviour of the neuron's transfer function.

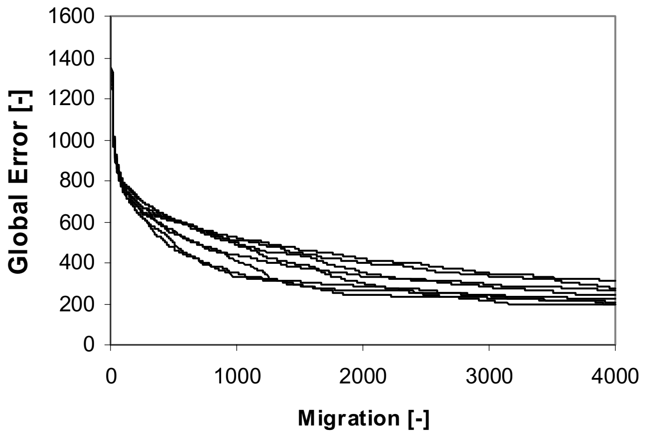


Fig. 4: An example of global error history progress

As can be seen in Figure 5, nearly the same platform efficiency as in the first test was reached. There are serverals parts of code which can be improved from the efficiency point of view. The fully artificial TimeCluster test shows probably the highest efficiency that can be reached at this stage of development. Remember, that as computation nodes the computers in study classrooms were used and performed simulations were interrupted by computer reboot caused students from time to time. Real applications using described parallel framework might show results similar to the neural network teaching test.

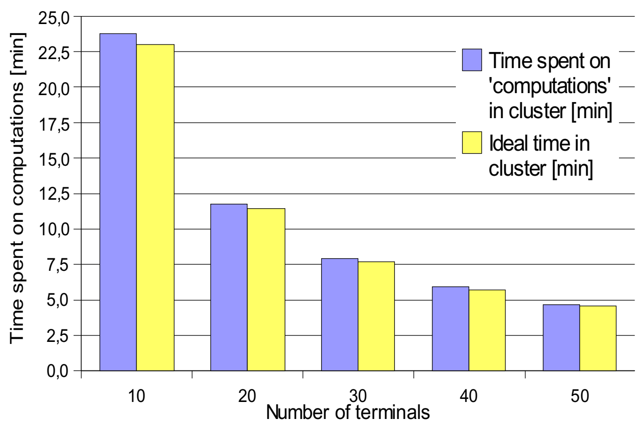


Fig. 5: Neural network taught by SOMA – ideal and real times

CONCLUSION

A platform for parallel computations was described. Results of efficiency tests demonstrated, that the framework is ready to use for distributed computing. Furthermore, the parallel version of SOMA was introduced and an example of its utilisation was shown. The future work will concentrate on practical applications of parallel SOMA and its improvement. Furhermore, a parallel version of Differential Evolution for the mentioned platform will be thoroughly tested.

ACKNOWLEDGEMENT

This work was supported by grant No. MSM 26500014 of the Ministry of Education of the Czech Republic and by grants of the Grant Agency of the Czech Republic GACR 102/03/0070.

REFERENCES

Bäck T., Schwefel H. P., 1993, An overview of evolutionary algorithms for parameter optimization. Evolutionary Computation Journal, No.1, 1-23.

Zelinka I., 2002, Artificial Intelligence in the Problems of Global Optimization (Czech Edition), BEN – technicka literatura, Praha 2002, ISBN 80-7300-069-5.

Zelinka I., 2004, SOMA – Self Organizing Migrating Algorithm",Chapter 7, 33 p. in: B.V. Babu, G. Onwubolu (eds), New Optimization Techniques in Engineering, Springer-Verlag, ISBN 3-540-20167X

[4] Bajaj R., Agrawal D. P., 2004, Improving Scheduling of Tasks in a Heterogenous Enviroment, IEEE Transactions on Parallel and Distributed Systems, vol. 15, No. 2, February 2004

[5] Bose N.K., Liang P. 1996, Neural Network Fundamentals with Graphs, Algorithms, and Applications, McGraw-Hill Series in Electrical and Computer Engineering, ISBN 0-07-006618-3, 1996

[6] Ka-Po Chow, Yu-Kwong Kwok, 2002, On Load Balancing For Distributed Multiagent Computing, IEEE Transactions on Parallel and Distributed Systems, vol. 13, No. 8, August 2002

[7] Masters T, 1993, Practical Neural Networks Recipes in C++, Academic Press, ISBN 0-12-479040-2, 1993

[8] Price K., 1999, An Introduction to Differential Evolution, in New Ideas in Optimization, D. Corne, M. Dorigo and F. Glover, Eds., s. 79–108, McGraw-Hill, London, UK, 1999. ISBN 007-709506-5

[9] Topcuoglu H., Hariri S., Min-You Wu 2002, Performance-Effective and Low-Complexity Task Scheduling For Heterogenous Computing, IEEE Transactions on Parallel and Distributed Systems, vol. 13, No. 3, March 2002

[10] Yang Y., Wang J., 2003, Nonblocking k-Fold Multicast Networks, IEEE Transactions on Parallel and Distributed Systems, vol. 14, No. 2, February 2003

AUTHOR BIOGRAPHIES

MIROSLAV ČERVENKA was born in Czech Republic, and went to the Tomas Bata University in Zlín, where he studied technical cybernetics and obtained his master degree in 2003. Now, he is a Ph.D. student at the same university. His e-mail address is `cervenka@ft.utb.cz`

IVAN ZELINKA was born in Czech Republic, and went to the Technical University of Brno, where he studied technical cybernetics and obtained his degree in 1995. He obtained Ph.D. degree in technical cybernetics in 2001 at Tomas Bata University in Zlín. Now he is associated professor (artificial intelligence, theory of information) and head of department. His e-mail address is `zelinka@ft.utb.cz` and his web-page can be found at `www.ft.utb.cz/people/zelinka/`