computing@computingonline.net
www.computingonline.net

ISSN 1727-6209
International Journal of Computing

RELIABLE AND EFFICIENT PARALLEL COMPUTING ON THE BASE OF MULTIAGENT SYSTEM

Aleksej Otwagin

United Institute of Informatics Problems, National Academy of Sciences of Belarus,
6 Surganov str., Minsk, 220012, Belarus. Email: forlelik@yahoo.com

Abstract: *Basic principles of reliable parallel computations are considered. The parallel program is represented as a graph computation schema, that executed by unreliable computing system with possible node faults. For organization and optimization of parallel processing a kind of multiagent architecture was used. The proposed solution uses the principles of runtime evolutionary optimization to increase performance characteristics.*

Keywords: *parallel processing, reliability, runtime optimization, multi-agent architecture, evolutionary algorithms.*

1. INTRODUCTION

The reliability of computing is the most important problem in many information systems, especially in case of large amount of interacting components and complicated architectures.

Increased requirements to data processing systems (necessity of guaranteed computation results in the conditions of active counteraction) are characteristic for military science, extreme situations, scale financial activity etc. In such applications unreliable realization of computations conducts to inevitable loss of the control over a situation that, eventually, leads to the spontaneous succession of events leading to inadmissible losses.

Computers with parallel architecture, and also the distributed computer networks (such as Internet/Intranet) are the most powerful systems for large data-intensive computation tasks. Therefore in such computing environments there are conditions not only for increase of computing speed with parallelization, but also for increase of their reliability with introduction at a scheduling stage a superfluous computing actions (duplication of resources for storage, transformation and transmission of data, and also carrying out of repeated actions of transformation and delivery of data). Thus quality of programs and computing processes is defined by a combination of criteria of parallelization speedup and reliability of calculations.

High complexity of the machine environment both in parallel computers and in the distributed computer networks become the formidable factor as various realization of programs differ by criterion of quality. Realizations of poor quality are easier for constructing, but they do not provide desirable advantages. For achieving of realizations of high quality the big additional expenses for optimization are necessary. The decision of this problem is very hard.

For the decision of large-scale problems of computing by means of the distributed computer networks the toolkit which will make possible direct programming of algorithms, accessible both to developers of algorithms, and experts is required. The creation of such toolkit for the synthesis of reliable parallel computing processes in the distributed man-machine environment containing unreliable components is an actual engineering task.

Multiprocessor systems allow applying of effective technology of component reservations which is directly realized by their structure.

The multiprocessor system contains p identical processors, and l processors can be considered as superfluous. The superfluous processor can replace any of the basic ones, that its duplication is realized. Each of the basic processors performs independent parts of overall computing algorithm. The system with duplication must be equipped with efficient reconfiguration subsystem. This subsystem transfers main processing functions on operable elements. Reconfiguration is realized with the intermediate layer of control and diagnostics software that is provided for checkpoint and restart of processes.

The multiagent systems [1, 2] are one of perspective technologies for realization such kind of intellectual control middleware. The separate subtasks solved during functioning of system agents

forms an overall system behavior that emerges from local interactions. Agents plan its own work and interaction so that to achieve the maximum benefit at the problem decision. When all agents achieve high efficiency of functioning then all system will solve whole problem more effectively.

Optimization of parallel computing system architecture to solved problem structure guarantees high real productivity of multiprocessing system for many classes of problems with any computational structure.

There are many examples of systems for parallel and distributed data processing [3-6]. These systems are different in used techniques and architectures, for example CORBA [6] or agent-based architecture [3].

Our contribution consists in development of a multiagent object-oriented framework, which is based on the MPI (Message Passing Interface standard) [7] for parallel computations. Another significant attribute of this framework consists in application of new hybrid algorithms for optimization of runtime scheduling. The realized framework uses special mechanisms for achieving reliable and effective computations in case of system components failures. The ability of online optimization allows applications to achieve high speed and multicomputer utilization.

The rest of the paper is organized as follows. Section 2 presents the model of developing parallel applications on distributed PC or supercomputer clusters. Section 3 describes the architecture of runtime parallel framework for optimization of computations. Section 4 presents the performance evaluation of proposed framework on experimental application.

2. A PARALLEL APPLICATION MODEL

The parallel processing of data assumes using of computer cluster [8]. While most clusters are homogeneous in real world, let's consider a case of a heterogeneous cluster that is more general. The heterogeneous cluster of computers consists from processors $P=\{p_1, p_2,..., p_m\}$, where p_i is an autonomous processor (also called node). Each processor p_i is weighted by w_i, which represents the time it takes to perform one unit of computation. The nodes in the heterogeneous cluster are connected by a high performance communication subsystem. Each communication link between computers p_i and p_j, denoted by l_{ij}, is weighted by s_{ij}, which indicates the time for transfer one unit of message data between p_i and p_j.

The main task of parallel computations consists in execution of computing operations set that is defined by program schema representing direct acyclic graph. The parallel applications, considered in this work, are used for processing of separate data objects, which are different in its types and processing schemas. A set of types $T=\{t_1,\ldots,t_n\}$ forms different scenarios for processing that are combined to one parallel program.

A processing system performs a set of processing operations $O=\{o_1,\ldots,o_k\}$. Each operation $o_k \in O$ is executed on a dedicated node of cluster P_i^k that has enough resources for executing of this operation. Each operation o_i is characterized by set of execution costs $w_{o_i}=\{w_{o_i}^1,\ldots,w_{o_i}^n\}$ for each data type, which represents the amount of computations while performing operation. Two operations for different data objects, which are performed on some fixed processor, cannot be executed in the same time, and two instances of one operation for different data objects also must be executed in different times or by different processors.

We represent parallel application in the form of Directed Acyclic Graph (DAG). DAG is represented as a tuple $G=(V,E,W,C)$, where:

V is a set of graph vertices $v_i \in V, 1 \le i \le N$. Each vertex is associated with processing operation from an operation set $O=\bigcup O_j$. A set of graph vertices represents decomposition of parallel program on the separated operations;

E is a set of graph edges $\{e_{i,j}=(v_i,v_j)\} \in E, i=\overline{1,N}, j=\overline{1,N}, i \ne j$. An edge represents a precedence relation between operations in program;

W is an operation cost matrix $W=\bigcup W_{o_i}^T$, where T indicates the type of processed object;

C is an edge cost set, where $c_{i,j} \in C$ determines the communication volume between two data processing operations, which is transferred by edge $e_{i,j} \in E$. We consider those operations, which are related and connected by the edge, use an identical data format for a predecessor output and a successor input. Thus the transfer costs for all types of processed objects are equal.

A design of parallel processing system consists in mapping of the program graph onto cluster topology. A parallel program is represented as a decomposition $((O),(P)) \longrightarrow \bigcup_{p \in P}(O_p)$, where $\forall O_k, O_p : O_k \ne O_p \supset O_k \bigcap O_p = \varnothing$. Each operation subset O_p can be placed on selected processor node.

The main purpose of parallel program is achieving of efficiency metrics (reduction of processing time, processing speedup, minimization

of resource requirements etc.). Let's denote as t_F^i the ending moment of operation o_i and the main objective of processing system will be a minimization of data object processing time

$$Z = \min(\sum_{i=1,n} t_F^i) \quad (1)$$

For evaluation of the schedule the simulation model is used, which is equal to real world parallel computation process.

An example of parallel program graph is presented in Fig.1, where each operation is denoted as *Oi* with defined cost (on the top), each operation process data objects with three different types *T1, T2, T3*. Cost of information transfers between operations pairs is equal for all data types. Some operations are strictly oriented on specific processor while others can be placed on each processor in cluster. If the operation cost for some type of data is equal to zero, then this operation must be skipped for selected type of data.

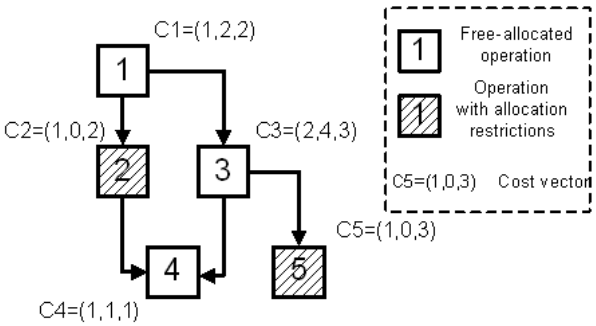

Fig. 1 – An example of program graph and denotation semantic

The main feature of systems for parallel processing that are considered in this paper is iterative computation mechanism. The task graph is applied to each data object from input stream according to the type of object and forms a cycle of iterative computations. The application is able to process a stream of data object with different size and order.

A main task of processing system is a planning and optimization of parallel program execution with simultaneous provision of reliable computations and guaranteed processing. There exist many algorithms of DAG scheduling that use various optimization techniques and heuristics [9-11]. However all static scheduling algorithms are constructed mostly for special graph topologies, or use special constraints, such as a zero communication time between nodes or an unbounded number of processors. Because of possible stochastic nature of input stream for many cases of parallel processing the static scheduling approach can not realize effective optimization.

Another perspective search techniques use evolutionary optimization. These techniques are based on such well-defined algorithms as a tabu search [12], simulated annealing, and genetic algorithms [13]. The most powerful is a genetic algorithm (GA) technique, and many algorithms are proposed in this field. However, the classical genetic algorithm is a blind search technique with extreme large search space. To speedup search procedure of genetic algorithms an algorithm of virtual associative network [14-16] can be used, and composition of these methods give us a kind of hybrid genetic algorithm (memetic algorithm [17, 18].

The algorithm of a virtual network is based on a concept of associations between the particular operations and dedicated agents. Each agent contains a vector of beliefs $B = \{b_{O_1}, b_{O_2}, \ldots b_{O_n}\}$, where each belief b_{O_k} determines a probability for selection of operation O_k for execution by this agent. Through runtime optimization agents could change its beliefs to adapt agent functioning process to general and local agent's objectives.

The operation O and a agent P, that is associated with corresponding processor, have linked by virtual link with force $B_{o,p} = b_{O_p} \cdot Z(o,p)$. A restriction matrix Z (O,P) can be introduced with following rules:

$Z(o,p) = 1$, if processor p allow execution of operation o (don't have resource restrictions);

$Z(o,p) = 0$, otherwise.

Restriction matrix is used in optimization procedures and prevents erroneous allocation of specified operations on some processors. The restrictions arises because of heterogeneous cluster structure and different operations requirements.

The algorithm uses associative memory for optimization, which is constructed on the base of association forces. This memory is learned by the accumulated experience, obtained in a solution search process. The algorithm is based on a GA representation of solutions in a form of population of chromosomes. Each chromosome represents a variant of program graph decomposition.

Proposed genetic algorithm uses a two-part chromosome representation, that allows simple realization of a genetic operators – mutation and crossover. Each chromosome C is a vector of number pairs, and chromosome gene $c_i \in C, c_i = (p,o)$. Index of gene corresponds to operation index in application DAG. The value of p

determines the number of processor, which executes this operation, and a value of o defines operation priority for scheduling. A priority is used when a processor has two or more ready operations for execution at present moment of time.

For genetic operators this chromosome is divided on two separate parts C^P and C^O, and each part is constructed using corresponding values from original chromosome (fig. 2).

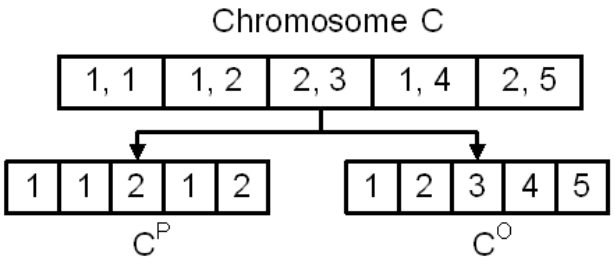

Fig. 2 - The chromosome structure.

The C^P part is used to determine an operation allocation, and C^O part is used for scheduling with priorities.

The mutation operator is performed by changing of operation allocation, and the corresponding gene is modified with new value for p, while o remains unchanged. The order mutation operator changes all C^O part by means of topological sorting of graph vertices. For example graph from figure 1 has the topological sort 1-3-2-5-4 with correct precedence relations between graph vertices. The C^P remains unchanged.

The crossover operator performs a classic one-point crossover for C^P parts of parent chromosomes. The C^O parts for child chromosomes are copied from C^O of parents.

These genetic operations exclude appearance of invalid allocations models for given graph. In this case the search procedure is reduced in time because the validation check for chromosomes is not needed now.

Each chromosome is evaluated by fitness function, which is based on a simulation model and satisfies the criterion (1). After the stage of evaluation and selection of a best solution candidate, the virtual network is learned by the positive experience. When the selected solution for some stage doesn't outperform previous best model, the virtual network is learned by previous experience. The learning procedure increases the association forces, which belongs to the best model.

The accumulation of experience allows the realization of a guided search in the solution space. This search is faster and gives better solutions at earliest stages of search. The size of population in the algorithm of the virtual associative network is smaller (5-10 chromosomes), than in classical GA (25-30 chromosomes), and requires less time for evaluation.

The virtual network algorithm introduces a new genetic operator – clusterization, which is performed with use of an experience from an associative memory. This operator allows a faster creation of stable schema in chromosomes, and thus an implementation of a genetic local search strategy. The clusterization operator means grouping of operations on processors with the strongest associations.

Each agent in multiagent system realizes a search procedure and selects a subset of operations, which are performed by this agent. The individual agent works on separate processor of cluster and uses a learning and planning policy, which is based on an association forces. The distributed version of virtual associative network allows all agents to create solution more effectively in cooperation. All agents uses unified architecture that is based on a parallel framework. Architecture of framework isolates the behavior logic that is dependent on the data processing schema, from the basic service code, that is common for all agents at modeling or application running.

3. THE FRAMEWORK ARCHITECTURE AND AGENT BEHAVIOR

The architecture of agent framework for parallel processing is presented in Fig. 3.

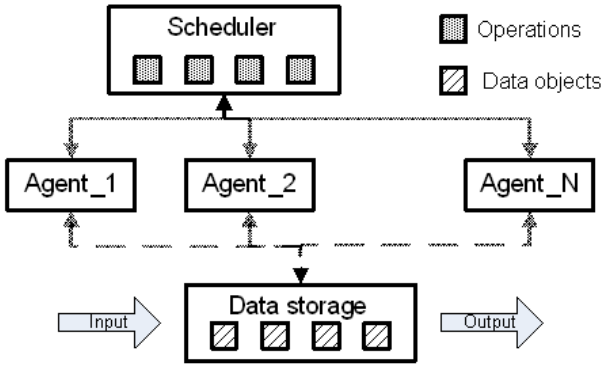

Fig. 3 – The architecture of parallel processing framework

The framework architecture is based on MPI and allows a fast communication between agents by means of internal MPI virtual machine. The MPI level of middleware realizes a name service for agents. Using of MPI replaces a MTS (Message Transport System) and DF (Directory Facilitator) services that are required by to FIPA specification for multiagent systems [19]. Basic agent structure is

presented on Fig. 4.

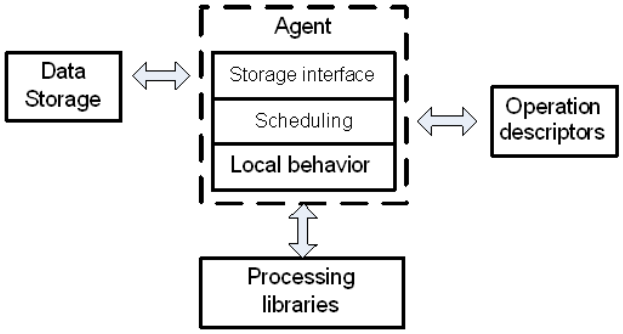

Fig.4. – The internal structure of agent

The input for multiagent system is a parallel program graph and a set of data objects that are different by their types. The parallel graph is represented as a XML file, which allows specifying all the characteristics of separate operations for all types of data objects.

Each agent performs parsing of whole graph and builds internal structures that are used for execution of specified operations with correct parameters for each object type. These operations are represented by descriptors, which are used by scheduler to control precedence relations and overall process.

The data object is represented by a data descriptor. This descriptor contains an identifier, type attributes and some additional information, for example, name of data file, which contains information for this object. When an operation requires additional data for processing, this descriptor must be extended for specified applications in appropriate way.

As the descriptors are transferred between processors of the parallel application, therefore the application code must contain serialization mechanisms. These mechanisms are realized for interaction with MPI facilities for messaging. The code is included in message transfer interface that is extensible and allows use of an alternative message transport systems.

Data objects are stored in a shared data storage that is realized as descriptor storage. Each descriptor is linked with universal container for storing of different data objects. The storage interface allows interaction with global storage for each agent in system. This interface has some facilities for object search, on-demand loading of remote objects and deletion of unused objects from storage. Each agent has a local copy of storage and uses it as a write-through cache.

The purpose of the scheduler interface is processing and scheduling control. It contains a special component, which is called a scheduler and makes decisions about the next processing operation that must be placed in descriptor queue. All descriptors of operations for processed objects are stored in operation descriptors storage that contains three sets of descriptors: ready, working and finished pools. The scheduler chooses the next processed operation from ready pool and sends its descriptor to an appropriate processor agent. After processing this descriptor is placed to the finished operations pool and information about next stage of processing is changed. The process repeats while the ready pool is not empty.

For reliability of computations there exists an intermediate working pool of descriptors. This pool was used to mark descriptors that are now executed by agents. When some agent is broken, then corresponding descriptor remains in this pool a long period of time. The scheduler periodically checks the descriptor state and moves these waiting descriptors back to ready pool. The descriptors then have a possibility to allocate on different working agent.

The agents are dynamically linked with a library of processing operations. Each processor executes operations that are specified by descriptors. The processor receives the descriptor from the coordinator, determines the next operation and executes it using the descriptor data. After completion of data processing, the descriptor is returned to scheduler. The processor works while stop instruction is not received.

Besides the process coordination, the runtime agents check system state and characteristics. These characteristics are collected and used for runtime optimization. The optimization is based on the measuring of data processing speed. When the input data changes its pattern significantly, the system must adapt to this situation. The adaptation performs reconfiguration of the operation subsets for all processor agents. The system tries to adapt to changed conditions and to achieve a high processing speed.

Agents use two different policies to choose of next operation from descriptor pool. First one consist in choosing operations on the base of agent's preferences. These preferences are formed in working process by the means of virtual associative network algorithm. Each agent have a vector of weights, and probability of selection of operation O for this agent A is:

$$\mathrm{P} = \frac{\omega_{\mathrm{O,A}}}{\sum_{\mathrm{i=1,O}} \omega_{\mathrm{i,A}}} \cdot Z(O, A) \cdot \qquad (2)$$

When an agent performs some operation and it's performance characteristics are increased, then corresponding operation weight is corrected

according to:

$$\omega_{O,A}(t+1) = \omega_{O,A}(t) + \alpha, \qquad (3)$$

where α is a learning coefficient.

The weights of remaining operations are corrected according to:

$$\omega_{O,A}(t+1) = \omega_{O,A}(t) - \alpha/(N-1), \qquad (3)$$

where N means overall amount of agents. An agent can choose from a subset of operations, taking first ready operation.

The second policy for agent is a greedy policy that consists in choosing of first ready operation from ready pool. This policy is introduced to eliminate a situation, when some descriptors are not chosen by long time. The greedy agents execute these operations and later they can choose this operation type as preferable. Each agent can switch between two scheduling policies when its local objective can not improve.

A local objective for agent is used for scheduling and interaction with other agents. Let the overall uptime of multiagent system is denoted by T units of time, and overall work time (time of operation execution) for agent A is denoted by T_A. The utility coefficient for this agent $W_A = T_A / T$, and a local objective for agent functioning is a minimization $Z_A = \min(W_A)$.

4. AN EXPERIMENTAL STUDY

For evaluating of proposed algorithms and a framework some experiments are performed. The experimental application for parallel processing of image data was used. All experiments have been done on the massively multiprocessor system K-1000, developed by United Institute of Informatics Problems.

In table the results of comparison for static decomposition schema and for dynamic optimization by the multiagent system are shown. The values shows relative improvement (in %) of processing time for dynamic optimization.

Table. The comparison of static and dynamic optimization

Processors count	Data objects count		
	20	40	80
2	0.78	2.15	4.73
4	1.49	4.41	5.19
8	3.06	4.83	5.46

The results shows, that the multiagent system outperforms the static data processing schema in case dynamic optimization of application structure.

5. CONCLUSION

An adaptive optimization improves image dataflow processing and brings a new level of intellectual behavior into systems. On the other hand, the modern technologies of optimization allow the minimization of expenses for design and evaluation of such systems. The suggested approach and framework will find their place at creation of modern dataflow processing systems for industrial applications.

The architecture of framework, based on an algorithmic skeleton approach, is suitable for many applications, which are distributed and use a graph representation. This framework can be extended by new operation sets for developing applications for another distributed processing problem areas.

6. REFERENCES

[1]. H.S. Nwana. Software Agents: An Overview. // The Knowledge Engineering Review. – 1996. – №11. – P. 205-244.

[2]. M. Wooldridge. Agents and software engineering // AI*IA Notizie. – 1998. - Vol. 11. - № 3. - P. 31-37.

[3]. D. Argiro, S. Kubica, M. Young, and S. Jorgensen. Khoros: An integrated development environment for scientific computing and visualization. Whitepaper, Khoral Research, Inc., 1999.

[4]. M. Zikos, E. Kaldoudi, S. Orphanoudakis. DIPE: A Distributed Environment for Medical Image Processing. *Proceedings of MIE'97*, Porto Carras, Sithonia, Greece, May 25-29, 1997, pp. 465-469.

[5]. M. Guld, B. Wein, D. Keysers, C. Thies, M. Kohnen, H. Schubert, and T. Lehmann, "A distributed architecture for content-based image retrieval in medical applications," in *Proceedings of the 2nd International Workshop on Pattern Recognition in Information Systems*, pp. 299–314, 2002.

[6]. J. Wickel, P. Alvarado, P. Dörfler, T. Krüger, and K.-F. Kraiss. Axiom — a modular visual object retrieval system. In M. Jarke, J. Koehler, and G. Lakemeyer, editors, *KI 2002: Advances in Artificial Intelligence*, LNAI 2479. Springer, 2002, p. 253–267.

[7]. W. Gropp, E. Lusk, and A. Skjellum Using MPI: Portable Parallel Programming with the Message Passing Interface. MIT Press, 1995.

[8]. K. Hwang, Z. Xu. Scalable Parallel Computing – Technology, Architecture, Programming. McGraw-Hill, USA, 1998.

[9]. B. S. Macey, A. Y. Zomaya. A performance evaluation of CP list scheduling heuristics for communication intensive task graphs. *In Proc. of IPPS/SPDP,* 1998, p. 538-541.

[10]. D. A. Menasce, D. Saha et al. Static and dynamic processor scheduling disciplines in heterogeneous parallel architecture. *Journal of Parallel and Distributed Computing*. Vol. 28, 1995. – pp. 1-18.

[11]. H. Oh, S. Ha. A Static Scheduling Heuristic for Heterogeneous Processors. *Second International EuroPar Conference Proceedings*, Vol II., Lyon, France, 1996, p. 573-577.

[12]. A. S. Porto, A. C. Ribeiro. A Tabu Search Approach to Task Scheduling on Heterogeneous Processors under Precedence Constraints. *International Journal of High-Speed Computing*, 2 (7), 1995, p. 45-71.

[13]. Z. Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. Second, Extended Edition. Springer-Verlag. 1994.

[14]. Y. M. Yufik, T. B. Sheridan. Virtual Networks: New framework for operator modeling and interface optimization in complex supervisory control systems // A Rev. Control, vol. 20, p. 179-195.

[15]. R. Kh. Sadykhov, A.V. Otwagin. Solution search algorithm of solution search for systems of parallel processing based on a virtual neural network model. *Automatic Control and Computer Science,* vol. 35 (1), 2001, Allerton Press Inc., New York, p. 25-33.

[16]. R. Kh. Sadykhov, A. V. Otwagin. Algorithm for optimization of parallel computation on the basis of genetic algorithms and model of a virtual network. I*n Proceedings of the International Workshop on Discrete-Event System Design DESDes'01*, Przytok, Poland, June 27-29, 2001, p.121-126.

[17]. R.W. Cheng, M. Gen. Parallel machine scheduling problems using Memetic Algorithms // Computers & Industrial Engineering. – Vol. 33. - № 3-4. – 1997. - P. 761-764.

[18]. N.J. Radcliffe. Formal memetic algorithms // Evolutionary Computing: AISB Workshop; ed. T.C. Fogarthy. - Springer Verlag, 1994. – P. 1-16.

[19]. S. Poslad, P. Buckle, R. Hadingham. Open Source, Standards and Scaleable Agencies. *International Workshop on Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, June 03-07, 2000, Manchester, UK, p.296-303.

This research is partially supported by Belarusian Republican Foundation of Fundamental Research, grant F08R-186.

Aleksej Otwagin *was graduated in Belarussian State University of Informatics and Radioelectronics (BSUIR) in 1998. He receives Ph.D. in Computer Sciences degree in 2007 and currently works as Assistant Professor in Computer Science Department at BSUIR. His scientific interests include parallel and distributed computing, evolutionary optimization and multi-agent architectures.*