New Generation Computing, 22(2004)177-186
Ohmsha, Ltd. and Springer-Verlag

A Grid-Oriented Genetic Algorithm Framework for Bioinformatics

Hiroaki IMADE, Ryohei MORISHITA, Isao ONO and Norihiko ONO
The University of Tokushima
2-1 Minamijosanjima, Tokushima,770-8506, Japan

Masahiro OKAMOTO
Kyushu University
6-10-1 Hakozaki, Higashi-ku, Fukuoka, 812-8581, Japan
hiro1121@is.tokushima-u.ac.jp

Received 15 June 2003
Revised manuscript received 17 November 2003

Abstract In this paper, we propose a framework for enabling for researchers of genetic algorithms (GAs) to easily develop GAs running on the Grid, named "Grid-Oriented Genetic algorithms (GOGAs)", and actually "Gridify" a GA for estimating genetic networks, which is being developed by our group, in order to examine the usability of the proposed GOGA framework. We also evaluate the scalability of the "Gridified" GA by applying it to a five-gene genetic network estimation problem on a grid testbed constructed in our laboratory.

Keywords: Genetic Algorithms, Grid, Framework, Genetic Networks, Grid-Oriented Genetic Algorithms.

§1 Introduction

In bioinformatics studies, it is important that researchers in life science, information science and bioinformatics collaborate with each other. By transparently using their resources such as data, power and services distributed in their laboratories, it is expected that large-scale problems that have been difficult to be handled by a single laboratory so far can be solved. Recently, to achieve such environment, some studies on grid computing have been made actively.[3,6] There are some grids for studying bioinformatics applications including North Carolina Bioinformatics Grid,[14] BioGrid[2] and Open Bioinformatics Grid.[9]

In bioinformatics studies such as estimating mutual interactions among genes and cell simulations, researchers need to find appropriate system structures and parameters of target systems that explain experimentally-observed data well. Generally, these processes are defined as optimization problems. These problems are very difficult because they are multidimensional, multimodal and non-linear problems and it takes a long time to evaluate a solution in these problems. The Genetic Algorithm (GA)[4] is an optimization method inspired by the evolution of living things. Because GAs not only show good performance on multidimensional, multimodal and non-linear problem spaces but also can be parallelized easily, they have been employed as optimization methods for solving these problems in many studies.[1,7,12,15,20,21] At present, most of these studies handle just small-scale problems by using a limited number of computation nodes in each laboratory independently. In order to handle larger-scale problems by using more computation nodes effectively, it is important that more researchers and more laboratories collaborate with each other. Therefore, GAs that can solve large-scale problems in a short time by effectively using a great number of computation nodes on a grid are expected to develop. In this paper, we call a GA running on a grid "Grid-Oriented Genetic Algorithm (GOGA)".

There are many studies on parallel GAs for reducing search time by using multiple computers.[5,8,11,18] However, most of these studies assumed the use of parallel computers, *e.g.* SMP computers, or PC clusters, which consist of homogeneous computation nodes and have high-speed and stable network. Grids are greatly different from parallel computers and PC clusters in that a grid consists of heterogeneous computation nodes and its network is low-speed and unstable. Most of existing parallel GAs cannot effectively work on a grid because asynchronous processes in heterogeneous environment, communication overhead, robustness and security are not considered. In order to develop GOGAs, GA researchers have to learn various domain knowledge on gird computing such as security, transferring data and invoking remote processes. Even if GA researchers use some middlewares such as Globus Tool Kit (GTK),[3] the researchers have to write complex source codes. Therefore, it is very difficult for ordinary GA researchers to begin GOGA studies at present.

This paper proposes a GOGA framework to easily develop GOGAs without domain knowledge on grid programming. In order to show that GAs used in bioinformatics can be easily "Gridified" by using the proposed framework, we try to "Gridify" a parallel GA for estimating genetic networks[13] as an example. Then, we examine the scalability of the proposed framework by applying the "Gridified" GA to a five-gene genetic network estimation problem on a grid testbed with three sites, which has been constructed in our laboratory.

§2 GOGA Framework

2.1 Requirements from A Viewpoint of GA Models

Generally, GAs can be categorized into the following two models:

The single population model is a model in which a single population is evolved

by genetic operators. Simple GA[4] and MGG[16] are its examples. Generally, its algorithm can be described as follows: 1) generate an initial population randomly (*generation of initial population*), 2) choose parents from the population (*selection for reproduction*), 3) generate kids by crossover and mutation (*generation of kids*), 4) calculate the evaluation values of the kids (*calculation of evaluation values*) 5) select the individuals surviving in the next generation (*selection for survival*), and 6) repeat the above steps from two to five until a terminating condition is satisfied. This model can be parallelized by assigning the population data and the processes other than *calculation of evaluation values* to one node, named *master*, and assigning the processes of *calculation of evaluation values* to multiple nodes, named *workers*.

The multiple population model is a model in which multiple sub-populations are evolved independently and interact with each other periodically. Island model[18] and DuDGA[8] are its examples. Generally, its algorithm can be described as follows: 1) generate multiple initial sub-populations (*generation of initial population*), 2) apply genetic operators to each sub-population (*independent evolution*) and 3) exchange information, such as the best individuals in each sub-population, among sub-populations periodically (*interactions among sub-populations*). This model can be parallelized by assigning each sub-population and the processes of *independent evolution* to each node, named *peer*.

In this paper, a persistent process on a node such as the process of *generation alternation* on a master and the process of *independent evolution* on each peer is called a *job*. A transient process invoked by other nodes such as the process of *calculation of evaluation values* and the process of *interactions among sub-populations* is called a *task*. A persistent data on a node such as a population on a master or a peer is called *session data*. A GOGA framework should provide some programming interfaces to easily define *jobs*, *tasks* and *session data*. A GOGA framework should also provide a function which jobs and tasks use to register any session data to a node, to refer them and to delete them from the node, a function which enables multiple jobs to operate cooperatively, a function which jobs use to request tasks to other nodes, and a function to easily realize hiding network latency.

2.2 Requirements from A Viewpoint of Grid Environment

In this paper, we assume a grid in which multiple sites connect to each other via the Internet and each site includes *global nodes* and *private nodes*, as shown in Fig. 1. A *global node* has a global IP and can be directly accessed from other sites. A *private node* has a global or private IP and cannot be directly accessed from other sites. Each private node can be directly accessed from the global nodes belonging to the same site.

A GOGA framework should satisfy the following requirements to work on such grid as described above: 1) the framework should have a powerful user authentication function and a data encryption function **(security)**, 2) the framework should have a function to notify errors on the network and other nodes to the user-defined code level and, then, to safely separate the troubles from the

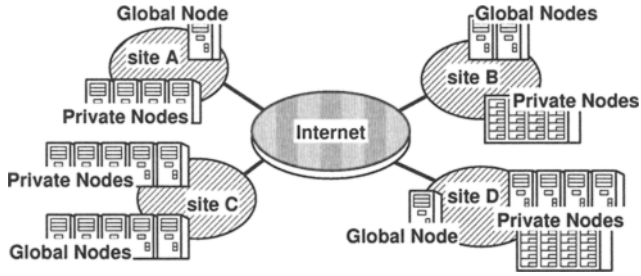

Fig. 1 Grid Environment

system, not stopping the whole system **(robustness)**, 3) the framework should have a function to add/delete nodes to/from the grid, not stopping the whole system**(flexibility)**, 4) the framework is desirable to run on as various platforms as possible **(portability)** and 5) the framework should provide functions to transfer any files, any tasks and any messages from any private node to another and to invoke any jobs on the other remote private nodes, and should have a single-sign-on function that allows users to login the grid by typing a pass phrase only once **(convenience to GOGA users)**.

2.3 Implementing GOGA Framework

[1] Programming language and middleware

We employ Java as a programming language from the view points of hiding network latency, robustness, flexibility and portability. Java is an object-oriented programming language that provides powerful multithread management, memory management, exception handling functions and good portability. We use Globus Tool Kit (GTK),[3] which is defacto standard for grid programming, as middleware. GTK provides functions for authenticating users based on X.509 certificates, enabling single-sign-on, invoking processes on remote global nodes, transferring files and standard input, output, or error messages to remote global nodes, and encrypting data transferred between global nodes. We also use CoG Tool Kit[10] to access GTK with Java.

[2] Architecture of GOGA framework

The GOGA framework consists of the following four programs:
Framework manager and ***job manager*** are programs running on a private node which a user uses as a terminal. Framework manager, first, invokes gateway managers on global nodes and local managers on private nodes. Next, job manager invokes specified jobs on specified local managers. Job manager transfers error messages from gateway managers and local managers to the user terminal and displays on the user console. These programs require only Java and CoG.
Gateway manager is a program running on more than one global node in each site. This program manages a database of nodes in the site, named *sitemap*, and relays tasks and messages between sites. The user can add some nodes to a sitemap

or delete ones from a sitemap by using framework manager. Communication between gateway managers is encrypted by SSL. This program requires Java, CoG, GTK and rsh/rcp clients.

Local manager is a program running on each private node. It manages jobs and session data and handles tasks received from other local managers. In local manager, the process of sending, receiving and performing tasks are assigned to independent threads, respectively. This allows a GOGA developer to easily realize hiding network latency and robustness by terminating only the corresponding threads at the time when some malfunctions occur. Standard-output messages from jobs and tasks are automatically recorded in a log file on the node. This program requires Java and rsh server.

[3] Programming interfaces

The GOGA framework provides the following classes to enable a GOGA developer to easily "Gridify" parallel GAs:

AbstractJob: A developer can define any jobs by extending *AbstractJob* and writing codes for the initialization and the job process. This class provides methods for sending and receiving tasks. Errors during requesting tasks are notified by exceptions to the user-defined codes of job processes.

AbstrackTask: A developer can define any tasks by extending *AbstractTask* and writing codes for the task process and the serialization/deserialization of necessary data to perform the task. Data necessary for task communication are held as a TaskHeader object and used by the framework.

Environment: By using *Environment*, jobs and tasks can register any jobs and objects as session data to the local manager, to refer them and to delete them.

[4] Behavior of GOGA framework

Framework manager, first, generates a certificate by receiving user's pass phrase and, then, performs authentication processes on global nodes on each site. Next, framework manager sends all necessary files to all nodes on the grid, and, after that, invokes gateway managers on global nodes and local managers on private nodes. Job manager, first, reads *boot job files*. A *boot job file* includes

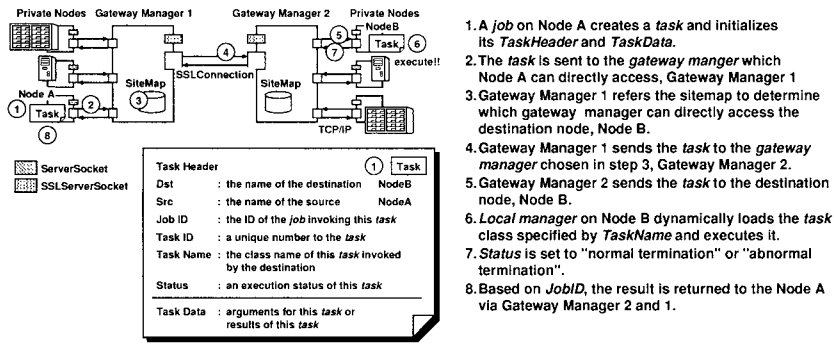

Fig. 2 Behavior of the Framework

the class name of a job, the node name where the job is to be invoked and some arguments for initialization and, then, invokes the jobs on the specified local managers. Figure 2 shows a typical behavior of the framework when a job requests a task.

§3 A Parallel GA for Estimating Genetic Networks and Its "Gridification"

3.1 A Parallel GA for Estimating Genetic Networks

This section briefly explains a parallel GA for estimating genetic networks, named "Parallel Network-Structure-Search Evolutionary Algorithm" (Parallel NSS-EA).[13] Parallel NSS-EA performs *structure optimization* on the master node and *parameter optimization* on multiple worker nodes as shown in Fig. 3. In *structure optimization*, the structures of genetic network are searched by a GA. The evaluation value of an individual in *structure optimization* is given by the evaluation value of the best individual found in *parameter optimization*. In *parameter optimization*, a real-coded GA searches the system parameters of network structures passed by *structure optimization*.

The master node has three kinds of threads cooperating with each other, named *main thread*, *generation-alternation thread* and *client thread*, and some data such as a population and an individual queue as shown in Fig. 3. *Main thread* initializes data such as the population and the individual queue. *Generation-alternation thread* handles the process of generation alternation for *structure optimization*. This thread makes kids and puts them to the individual queue. *Client thread* communicates with a worker. Multiple client threads run on the master node. This thread, first, initializes the corresponding worker and, then, repeats the processes of getting the predetermined number of kids from the individual queue and sending them to the corresponding worker.

A worker node, first, initializes itself by using data received from the corresponding client thread on the master node. Then, the worker repeats the following procedures: 1) receive a request for evaluating individuals, 2) perform *parameter optimization* of the individuals and 3) return the evaluation values obtained in *parameter optimization*.

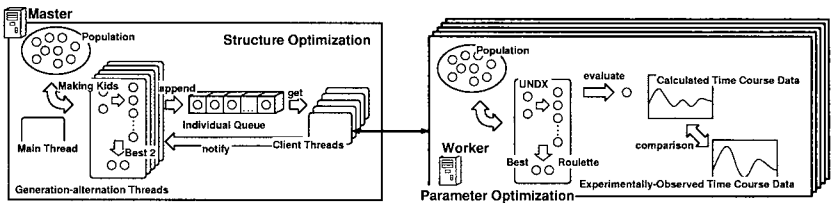

Fig. 3 Parallel NSS-EA

3.2 "Gridifying" the Parallel GA for Estimating Genetic Networks

In this section, we try to "Gridify" parallel NSS-EA by using the proposed

GOGA framework and show that parallel GAs can be easily "Gridified" by implementing threads and requests to a worker in parallel GAs as jobs and tasks of the GOGA framework, respectively, as described below.

In this paper, main thread, generation-alternation thread and client thread are implemented as jobs, named *MasterMainJob*, *GenerationAlternationJob* and *ClientJob*, respectively, by extending *AbstractJob* of the GOGA framework. Requests for initialization and parameter optimization on a worker are implemented as tasks, named *InitWorkerTask* and *EvalTask*, respectively, by extending *AbstractTask* of the GOGA framework. The population and the individual queue are registered as session data. *MasterMainJob* generates a population and an individual queue, registers them to *Environment* as session data, and waits until the time when the number of generations reaches the finish generation. *GenerationAlternationJob* gets the population and the individual queue from *Environment*, and performs the processes of generation alternation. *ClientJob* gets the individual queue from *Environment*, sends *InitWorkerTask* to the corresponding worker to initialize the worker, and repeats the processes of getting the predetermined number of kids from the individual queue, packing them into *EvalTask* and sending the task to the worker. *InitWorkerTask* initializes the worker by registering the system parameters and time course data for *parameter optimization* as *session data*. *EvalTask* performs *parameter optimization* by using the session data registered by *InitWorkerTask*.

§4 Performance Test of GOGA Framework

To evaluate the scalability of the proposed GOGA framework, we applied the "Gridified" parallel NSS-EA to a five-gene genetic network estimation problem on the grid testbed as shown in Fig. 4. We measured the time required for 200 iterations when the number of workers is 1, 10, 20, ..., 100. Fig. 5 (left) shows the number of workers versus the time required for 200 generations. Fig. 5 (right) shows the number of workers versus the speed up rate, where the speed of one worker is supposed to be 1. The average time required for a single EvalTask to be accomplished on a worker is distributed between one and three second. The average size of transmission data is about 200 bytes. As shown in Fig. 5 (right), the GOGA framework shows good scalability. In this experiments, we also confirmed that GOGA framework notified errors to the user-level code by exceptions when some malfunctions occurred in any worker.

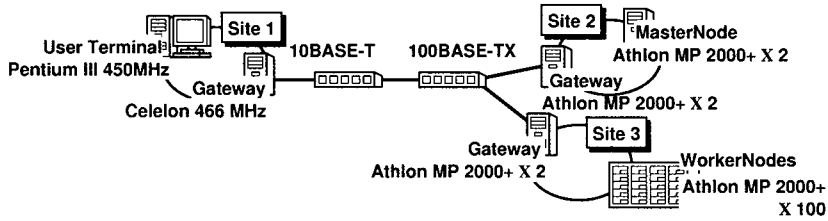

Fig. 4 Grid Testbed

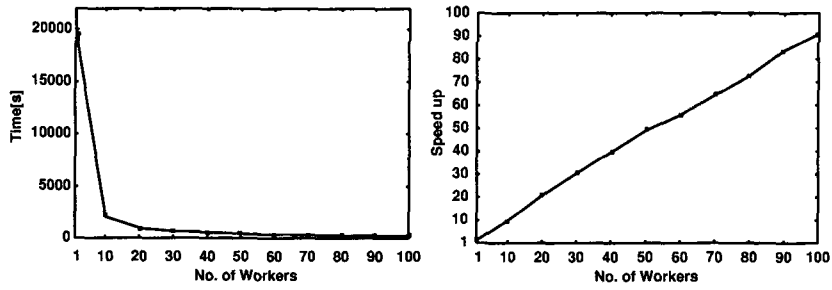

Fig. 5 Time Required for 200 Generations (left) and Speed Up Rate (right)

§5 Conclusion

In this paper, we discussed the requirements for GOGA framework and proposed an implementation of the GOGA framework. We tried to "Gridify" a parallel GA for estimating genetic networks. As the result, we confirmed that we can easily "Gridify" parallel GAs by using the proposed GOGA framework. We also found that the proposed GOGA framework showed good scalability.

For future work, we will evaluate the performance of the framework on Open Bioinformatics Grid where the network latency is very large. We will try to "Gridify" other existing GA models including multiple population models and evaluate the performance. We also have a plan to develop a new GOGA running efficiently on the grid. We have to compare the features and performance of the proposed GOGA framework with those of EVOLVE/G.[19] We will release the GOGA framework on Open Bioinformatics Grid in the near future.

Acknowledgements

This work was partially supported by the Grants-in-Aid for Scientific Research on Priority Areas, "Genome Information Sciences" (No.12208008) from the Ministry of Education, Culture, Sports, Science and Technology in Japan.

References

1) Ando, S. and Iba, H., "Inference of Gene Regulatory Model by Genetic Algorithms," *Proc. CEC 2001*, 2001.

2) *BioGrid*, http://www.biogrid.jp

3) Foster, I. and Kasseleman, C., "Globus: A Meracomputing Infrastructure Toolkit," *IJSA*, 1997.

4) Goldberg, D. E., "Genetic Algorithms in Search, Optimization and Machine Learning," *Addison-Wesley*, 1989.

5) Gorges-Scheluter, M., "ASPARAGOS An Asynchronous Parallel Genetic Optimization Strategy," *Proc. 3rd ICGA*, pp. 422–427, 1989.

6) Grimshaw, A. S., Wulf, W. A. and the Legion team, "The Legion Vision of World Wide Virtual Computer," *Communications of the ACM*, 1997.

7) Hatakeyama, M., Kimura, S., Naka, T., Kawasaki, T., Yumoto, N., Ichikawa, M., Kim, J., Saito, K., Saeki, M., Sirouzu, M., Yokoyama, S. and Konogaya A., "A Computational Model on the Modulation of MAPK and Akt Pathways in Heregulin-induced ErbB Signaling," Journal of Biochemistry, 373, 2, pp. 451–463, 2003.

8) Hiroyasu, T., Miki, M., Hamasaki, M. and Tanimura, Y., "A New Model of Distributed Genetic Algorithm for Cluster Systems, Dual Individual DGA," *Proc. PDPTA '2000*, 1, pp. 477–483, 2000.

9) Konishi, F., Fukuzaki, A., Satou, K., Yamamoto, T., Defago, X. and Konagaya, A., "OBIGrid: A New Computing Platform for Bioinformatic," *Genome Informatics*, *13*, pp. 484–485, 2002.

10) Laszewski, G., Foster, I., Gawor, J., Smith, W. and Tuecke, S., "CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids," in *ACM 2000 Java Grande Conf.*, pp. 97–106, 2000.

11) Lee, C., Park, K., Kim, J., "Hybrid Parallel Evolutinary Algorithms for Constrained Optimization Utilizing PC Clustering," *Proc. CEC 2001*, *2*, pp. 1436–1441, 2001.

12) Miyoshi, F., Nakayama, Y. and Tomita, M., "Estimation of Genetic Networks of the Circadian Rhythm in Cyanobacterium Using the E-Cell System," *Genome Informatics*, *12*, pp. 308–309, 2001.

13) Morishita, R., Imade, H., Ono, I., Ono, N. and Okamoto, M., "Finding Multiple Solutions Based on An Evolutionary Algorithms for Inference of Genetic Network by S-system," to be published in *Proc. CEC 2003*, 2003.

14) *North Carolina Bioinformatics Grid*, http://www.ncbiogrid.org.

15) Sakamoto, E., and Iba, H., "Inferring a System of Differential Equations for a Gene Regulatory Network by Using Genetic Programming," *Proc. CEC 2001*, 2001.

16) Sato, H., Yamamura, M. and Kobayashi, S., "Minimal Generation Gap Model for Gas Considering both Exploration and Exploitation," *Proc. IIZUKA '96*, pp. 494–497, 1996.

17) Savageau, M. A., "Biochemical Systems Analysis : A Stuff of Function and Design in Molecular Biology," Addison-Wesley, Reading, 1976.

18) Tanese, R., "Distributed Genetic Algorithms," *Proc. 3rd ICGA*, pp. 434–439, 1989.

19) Tanimura, Y., Hiroyasu, T. Miki, M. and Aoi, K., "The System for Evolutionary Computing on the Computational Grid," *IASTED 14th Intl. Conf. on Parallel and Distributed Computing and Systems*, pp. 39–44, 2002.

20) Tominaga, D., Koga, N. and Okamoto M., "Efficient Numerical Optimization Algorithm Based on Genetic Algorithm for Inverse Problem," *Proc. GECCO 2000*, pp. 251–258, 2000.

21) Ueda, T., Koga, N., Ono, I. and Okamoto M., "Efficient Numerical Optimization Technique Based on a Real-coded Genetic Algorithm for Inverse Problem," *Proc. AROB '02*, 2002.

Hiroaki Imade: He received his B.S. degree in the department of engineering from The University of Tokushima, Tokushima, Japan, in 2001. He received the M.S. degree in information systems from the Graduate School of Engineering, The University of Tokushima in 2003. He is now in Doctoral Course of Graduate School of Engineering, The University of Tokushima. His research interests include evolutionary computation. He currently researches a framework to easily develop the GOGA models which efficiently work on the grid.

Ryohei Morishita: He received his B.S. degree in the department of engineering from The University of Tokushima, Tokushima, Japan, in 2002. He is now in Master Course of Graduate School of Engineering, The University of Tokushima, Tokushima. His research interest is evolutionary computation. He currently researches GA for estimating genetic networks.

Isao Ono, Ph.D.: He received his B.S. degree from the Department of Control Engineering, Tokyo Institute of Technology, Tokyo, Japan, in 1994. He received Ph.D. of Engineering at Tokyo Institute of Technology, Yokohama, in 1997. He worked as a Research Fellow from 1997 to 1998 at Tokyo Institute of Technology, and at University of Tokushima, Tokushima, Japan, in 1998. He worked as a Lecturer from 1998 to 2001 at University of Tokushima. He is now Associate Professor at University of Tokushima. His research interests include evolutionary computation, scheduling, function optimization, optical design and bioinformatics. He is a member of JSAI, SCI, IPSJ and OSJ.

Norihiko Ono, Ph.D.: He received his B.S. M.S. and Ph.D. of Engineering in 1979, 1981 and 1986, respectively, from Tokyo Institute of Technology. From 1986 to 1989, he was Research Associate at Faculty of Engineering, Hiroshima University. From 1989 to 1997, he was an associate professor at Faculty of Engineering, University of Tokushima. He was promoted to Professor in the Department of Information Science and Intelligent Systems in 1997. His current research interests include learning in multi-agent systems, autonomous agents, reinforcement learning and evolutionary algorithms.

Masahiro Okamoto, Ph.D.: He is currently Professor of Graduate School of Systems Life Sciences, Kyushu University, Japan. He received his Ph.D. degree in Biochemistry from Kyushu University in 1981. His major research field is nonlinear numerical optimization and systems biology. His current research interests cover system identification of nonlinear complex systems by using evolutional computer algorithm of optimization, development of integrated simulator for analyzing nonlinear dynamics and design of fault-tolerant routing network by mimicking metabolic control system. He has more than 90 peer reviewed publications.