Vol.2011-MPS-84 No.6
2011/7/18

A Framework for Genetic Algorithms in Parallel Environments

Tomoyuki HIROYASU,[†1] Ryosuke YAMANAKA,[†2]
Masato YOSHIMI[†3] and Mitsunori MIKI[†3]

In this research, we developed a framework to execute genetic algorithms (GA) in various parallel environments. GA researchers can prepare implementations of GA operators and fitness functions using this framework. We have prepared several types of communication library in various parallel environments. Combining GA implementations and our libraries, GA researchers can benefit from parallel processing without requiring deep knowledge of different parallel architectures. In the proposed framework, the GA model is restricted to a micro-grained model. In this paper, parallel libraries for a Windows cluster environment, multi-core CPU environment, and GPGPU environment are described. A simple GA was implemented with the proposed framework. Computational performance is also discussed through numerical examples.

1. Introduction

Recently, several types of parallel architecture have come into wide use. For example, calculation with multi-core CPU which more than four cores is not unusual. General purposed GPU becomes also easy to use. In Japan, some of the supercomputing centers are open for researchers to use high-end computational resources. We can use the Earth Simulators and will be able to use the next-generation Keisoku supercomputer. However, these parallel architectures have the different configurations. Thus, even when we wish to use the same algorithms, it is necessary to prepare different implementation codes suitable for different parallel architectures. This places a heavy burden on algorithm researchers, because in-depth knowledge of the different parallel architectures is required to run their implementation codes efficiently on parallel machines.

GA is a type of optimization algorithm with multipoint search[1)]. GA may find the optimum point even when the landscape of the objective function has multiple peaks. However, GA requires much iteration to find the optimum. This results in high calculation cost. As GA is a multipoint search algorithm, it implicitly has several types of parallelism[2)3)4)5)]. Thus, several types of research regarding parallelization of GAs are existed. Ono et al[6)], introduced the GA model and implementation parallel models of GA should be clarified. As there is parallelism in the GA itself, parallel GA can be performed even on a single process. We call this the logical parallel model. On the other hand, because GA has multiple search points, a single model can be implemented on parallel computers. In this case, an implementation parallel model should be prepared.

In most GA research, these logical and implementation parallel models are not distinguished clearly and are often the same[7)8)9)]. When the logical model is closely related to the implementation model, GA users should have deep knowledge of the parallel architectures on which their parallel GAs are running. At the same time, as the logical model and implementation model are closely related, different parallel codes are required for different parallel machines. Therefore, it would be of great benefit if GA users were not required to have such deep knowledge of novel parallel architectures to run their GAs in parallel.

Here, we propose a parallel environment framework for GA that adopts the micro-grained model as an implementation model. GA researchers prepare the implementations of GA operators and fitness functions using the proposed framework. We are preparing parallel communication libraries for this framework. Using GA implementations and these libraries, GA users can derive efficient parallel GA codes without requiring specific knowledge regarding to parallel architectures. Thus, the proposed framework improves the productivity of GA users.

We constructed a system for Windows cluster with Windows Communication Foundation (WCF) in C# with multithreading in C language for three types of parallel environment, i.e., cluster, multi-core CPU, and GPU. In addition, we verified the system through the preliminary experiments.

2. Genetic Algorithm

2.1 Overview

The GA is an optimization algorithm that mimics natural evolution with varietion and adaptation to the environment. In evolution processes in nature, an

†1 Department of Life and Medical Sciences, Doshisha University
†2 Graduate School of Engineering, Doshisha University
†3 Department of Science and Engineering, Doshisha University

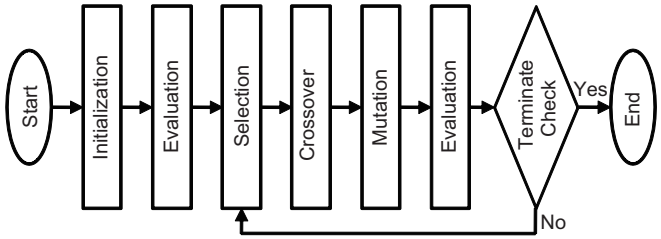

Fig. 1 Flowchart of GA.

individual that is better adapted to the environment among a group of individuals forming a certain generation survives at a higher rate, and leaves offspring to the next generation. In the GA concept, the computer finds an individual that is better adapted to the environment, or a solution that yields an optimum value to an evaluation function, by modeling the mechanism of biological evolution. **Fig. 1** shows a typical flowchart of GA.

The GA applies genetic operations, crossover and mutation, to each individual in the population to produce new individuals. These individuals are evaluated and the GA selects superior individuals for the next generation. The GA searches for a solution by repeating this series of operations until the termination condition is met. The evaluation time is the same as the number of populations. GAs have good performance in parallel environments because they have data-level parallelism.

2.2 Parallel Model of GA

The GA is able to parallelized because it searches multiple points and repeats sampling. Parallel models of GA can be divided into coarse-grained and micro-grained models

2.2.1 Coarse-grained model

The coarse-grained model is generally called a distributed population model. This model splits the population into multiple subpopulations, which are then searched. Therefore, several individuals in several subpopulations are moved into other subpopulations. This operation is called the migration. **Fig. 2** shows the flow of the coarse-grained model. This model uses computational resources effectively, because it connects to computational nodes only during migration. In addition, this model changes performance of the search compared to a serial algorithm.

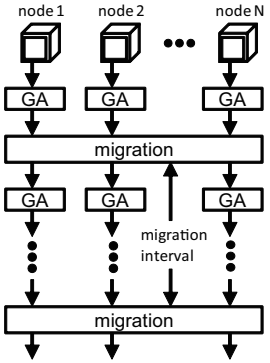

Fig. 2 Coarse-grained model.

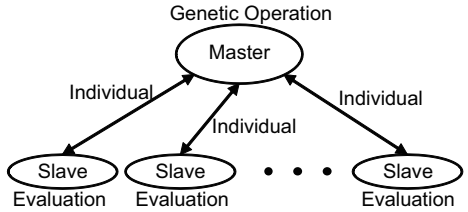

Fig. 3 Micro-grained model.

2.2.2 Micro-grained model

Evaluations account for a large share of total execution time in complex of objective problems. The micro-grained model is based on the general concept of parallelization. This model is a master-slave model. A master processor executes other genetic operations besides evaluation.

Evaluations are executed by slave processors. A master processor sends individuals that should be evaluated. Slave processors evaluate these individual, and return them to the master processor. **Fig. 3** shows the flow of micro-grained model. This model shows inferior parallelization performance compared to the coarse-grained model, because it must have many connections and the master processor uses a CPU. In addition, this model does not alter the search performance compared to a serial algorithm.

3. Proposed Framework

3.1 Background

In conventional parallel GA research, the proposed GAs are often fully connected to the particular parallel environment. In these cases, the proposed GAs cannot be used in the different parallel environments. Advanced programming techniques are required to use computational resources with GPGPU and many cores. As new architectures appear, more GA users will be required to make implementations for them. This represents a burden on GA users. The coarse-

grained model has important differences from the micro-grained model. The coarse-grained model has few connections and can use parallel environments. However, when this model is adopted, the designed GA and the search performance are changed. The micro-grained model, on the other hand, does not alter the search performance are changed. At present, users adopt the coarse-grained model to make effective use of parallel environments. Therefore, GA development is limited.

3.2 Requirements

To overcome the disadvantages discussed in the previous section, a framework is proposed with the following requirements:

- Systematization of parallel models
- Adoption of micro-grained model
- Standardized interface

These requirements are described in more detail below.

3.2.1 Systematization of Parallel Models

There are two aims of parallelization in GA. The first is to parallelize algorithms for improved search performance. The second is to parallelize the implementation to reduce computational time. We must declare two parallelization that parallel implementation does not confine algorithms of GA. We define them as a logical model and an implementation model. **Fig. 4** shows a GA that adopts the island model as a logical model and serial model as an implementation model. **Fig. 5** shows a GA that adopts the island model as both the logical model and as the implementation model. The logical model searches in parallel; however, it can also be implemented in serial. In addition, the logical model is confined by the limits of implementation model.

3.2.2 Adoption of Micro-grained Model

It is necessary for GA users to use an arbitrary algorithm with which any and all GAs can benefit from parallel processing. The micro-grained model is adopted as the implementation model. Therefore, the GA can be implemented without changing the logical model.

3.2.3 Standardized Interface

To reduce the burden on GA users, it is necessary that they should be able to use any and all parallel environments with a common interface.

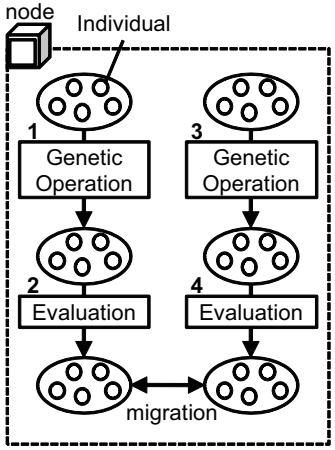

Fig. 4 Island model with serial implementation.

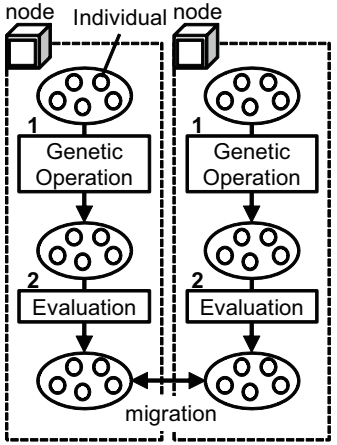

Fig. 5 Island model with parallel implementation.

3.3 Overview of Proposed Framework

The purpose of the proposed framework is to allow GA users to perform parallel processing with the micro-grained model as an implementation model of GA, without programming techniques for parallel processing. **Fig. 6** shows an overview of the framework. The framework introduces the concept of the GA Pool as the interface. GA users throw individuals into the GA Pool. Thus, they are able to get evaluated individuals from the GA Pool. The GA Pool evaluates thrown individuals with parallel environments. The framework supplies an implementation for use of parallel environments. GA users can construct an arbitrary logical model and implement GA operations besides evaluation part. They implement the evaluation part with the prepared template. This template has arguments and return value of the function of the evaluation. This template hides implementation of a particular connection and scheduling for the task of evaluation from the GA user. Thus, GA users can construct GAs adapted to parallel environments without requiring knowledge regarding connections and scheduling jobs.

3.4 GA Pool

The GA Pool is composed of two queues as shown in **Fig. 7**. These queues are the throw queue, which puts thrown individuals, and the get queue, which

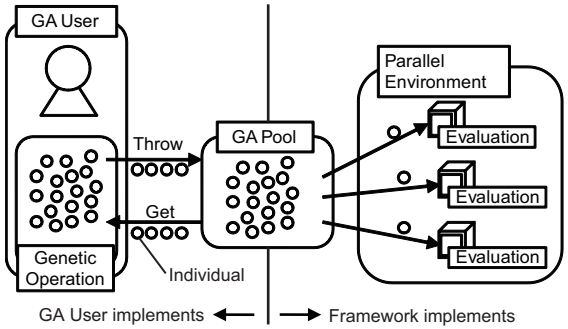

Fig. 6 Concept of the framework.

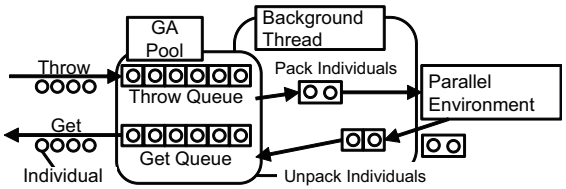

Fig. 7 Constitution and function of GA Pool.

puts evaluated individuals. When individuals are put, a thread that monitors the throw queue sends individuals to computational resources and calculates the evaluation. The thread executes connections adapted to the architecture of the computational resources in the parallel environment. When the thread puts evaluated individuals into the get queue, it unpacks them one by one.

4. Evaluation

4.1 Confirmation of Parallelism with Framework

This section confirms the parallelism of GAs constructed with the proposed framework in a cluster environment.

4.1.1 Computational Environments

A Windows cluster running Windows HPC Server was used as a distributed memory environment. The communication infrastructure was Windows Communication Foundation (WCF)[10] with C#, and we used a novel parallel model different from MPI. This parallel model is based on Service Oriented Architecture (SOA)[11][12][13]. Only function implementation is located on the computational nodes as slaves. The client machine acting as a master calls the function to control jobs interactively. WCF does not share sources between master and slave processors. Therefore, it has the good expandability. In addition, WCF adapted the micro-grained model such that the slaves execute evaluation only, because slave processors have only the function evaluation. The Windows cluster is able to view a core as a computational resource. This section discusses confirmation of the parallelism of GAs constructed with the proposed framework with 16 cores on 2 machines, as shown in **Table 1**. **Table 2** shows the parameters of the GA used in this section.

4.1.2 Results

Fig. 8 shows the relation between the number of computational nodes and execution time. **Fig. 9** shows pseudo-code of a simple GA constructed with the proposed framework. Lines 4, 8, 11, and 20 in Fig. 9 are descriptions for using the framework. GA users add only four descriptions and can reduce the execution time by increasing the calculation resources as shown in Fig. 8.

4.2 Verification of connection performance with along to data size

This section evaluates the connection performance of a cluster, multi-core CPU, and GPU. We verify the influences of data volume and number of connections on execution time with changing numbers of individuals in a connection. There are several parameters in algorithms and parallel libraries, and these parameters should be tuned optimally to achieve high parallel efficiency. In a distributed memory environment, communication overheads are large. In a shared memory environment, it is necessary to consider the limits of memory based on the processor architecture. Here, we use a system that controls data volume and number of connections when the master processor communicates with the slave processors. In particular, the system controls the number of individuals in a connection as shown in Fig. 7. During the simulation, the best data volume and number of

Table 1 Architecture of a node on the Windows cluster.

OS	Windows HPC Server 2008
Memory	8 GB
CPU	AMD Opteron 2536 2.3 Hz(Quad) × 2

Table 2 Parameters of GA.

Parameter	Value
Population Size	64
Gene Length	41
Max Generation	32
Optimization Problem	HRE[14]
Logical Model	Simple GA[1]

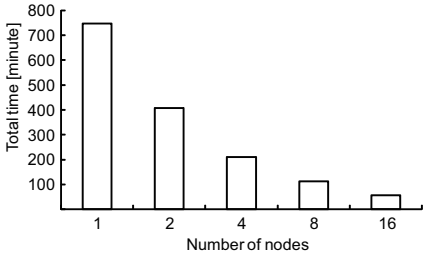

Fig. 8 Relation between number of computational nodes and execution time.

```
 1: // initialization of population
 2: InitPopulation();
 3: // initialization of framework
 4: Initialize(POPULATION_SIZE, MAX_GENERATION);
 5: for (int i = 1; i <= MAX_GENERATION; i++) {
 6:     for (int j = 0; j < POPULATION_SIZE; j++)
 7:         // throw individuals to GA Pool
 8:         Throw(individual);
 9:     for (int j = 0; j < POPULATION_SIZE; j++)
10:         // get individuals from GA Pool
11:         individual = Get();
12:     // selection
13:     population = selection(population);
14:     // crossover
15:     crossover(population);
16:     // mutation
17:     mutation(population);
18: }
19: // Finalization of framework
20: Finalize();
```

Fig. 9 SimpleGA constructed with the proposed framework

connections can be determined dynamically.

4.2.1 Environments

The cluster environment is the same as the system used in section 4.1. A multi-core CPU and GPU are used as shared memory environments. We used C language on multi-core CPU and CUDA with NVIDIA on GPU. In shared memory environments, multiple threads in relation to the number of cores are generated for parallel processing. GA users need not send a program of the evaluation module to computational resources. We use computational resources to compare each environment. One node is used for a Windows cluster and one thread is used for multi-core CPU and GPU.

Tables 3 and **4** show the specifications of multi-core CPU and GPU.

Table 5 shows the parameters of the GA used in this evaluation. A total 512 data (4000 bytes each) are sent to the computational resources, because the population size is 512 and the data volume of an individual is 4000 bytes. Power-of-two data are sent, and we record the execution time.

4.2.2 Results

Figures 10, **11**, and **12** describe the relation between data volume in a connection and execution time in each parallel environment. The values in these graphs are the medians of 100 independent trials. As show in each figure, the data volume was confirmed to affect the execution time. In addition, we confirmed that the optimal data volume that reduces the running time to the greatest extent is different for each architecture. The optimal values are 64 × 4000 bytes for the cluster, 8 × 4000 bytes for the multi-core CPU, and 16 × 4000 bytes for the GPU is.

Table 3 Architecture of node with multi-core CPU.

OS	Debian 5.0.8
Memory	16 GB
CPU	AMD Opteron 2423 2.0 GHz(Six-Core) × 2

Table 4 Architecture of node with GPU.

OS	CentOS
Memory	16 GB
CPU	AMD Opteron 2356 2.3 GHz(Quad) × 2
GPU Memory	512 MB
GPU	NVIDIA GeForceGTX250 1.84 GHz(128-Core)

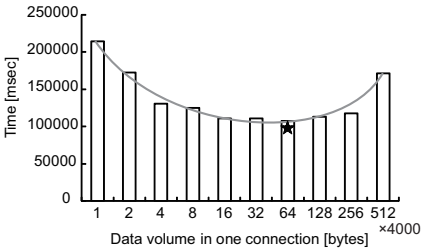

Fig. 10 Relation between data volume in one connection and execute time on Windows cluster.

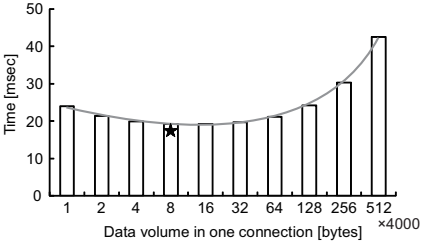

Fig. 11 Relation between data volume in one connection and execute time on multi-core CPU.

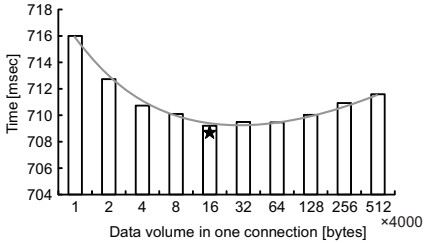

Fig. 12 Relation between data volume in one connection and execute time on GPU.

5. Discussion

As shown in Figures 10, 11 and 12, it was confirmed that the minimum execution time according to data volume was different between the cluster, multi-core

Table 5 Parameters of GA.

Population Size	512
Gene Length	1000
Data Type of a Gene	int (4 bytes)

Table 6 Differences between max and min times.

Environment	Difference [msec]
Cluster	107170
Multi-core CPU	23.1068
GPU	3.52358

CPU, and GPU environments. **Table 6** shows the differences between the max and min values of execution time in each environment. As shown in this table, the difference is smallest in GPU and the largest in cluster. This suggests that tuning has a greater effect in distributed memory environments. However, the tuning affects are also different even for shared memory environments. This evaluation used only homogeneous parallel environments. However, these results show that changing data volume has a greater influence in heterogeneous parallel environments.

6. Conclusions and Future Work

In this paper, we proposed a framework for GAs in parallel environments. GA researchers can prepare implementations of GA operators and fitness functions using this framework. We have prepared several types of communication library for use in various parallel environments. Combining the GA implementations and our libraries, GA researchers can benefit from parallel processing without requiring deep knowledge regarding parallel architectures. In the proposed framework, the GA model is restricted to a micro-grained model. In this paper, parallel libraries for Windows cluster environment, multi-core CPU environment, and GPGPU environment were prepared.

For the Windows cluster, parallel communication libraries were prepared with WCF and C#. Using the libraries and the framework, GA researchers can implement the parallel processing part with only four descriptions. In addition, we verified data volumes and number of connections on the Windows cluster, multi-core CPU, and GPU with a system that changes the number of individuals in a connection. The results indicated that the best number of individuals in a connection differs according to the architecture.

In future work, a mechanism to find the best number of individuals and to tune it dynamically will be implemented in the libraries. In addition, we will also attempt to prepare other parallel libraries for other parallel architectures.

References

1) Goldberg, D.E.: *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley (1989).
2) Starkweather, T., Whitley, D. and Mathimas, K.: *Optimization using Distributed Genetic Algorithms*, Parallel Problem Solving form Nature (1991).
3) Mühlenbein, H.: Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization, *Parallelism, Learning, Evolution*, Lecture Notes in Computer Science, Vol.565, Springer Berlin / Heidelberg, pp.398–406 (1991).
4) Theodore, C.: The Distributed Genetic Algorithm Revisited, *Proc.6th International Conf. Genetic Algorithms*, pp.114–121 (1995).
5) Miki, M., Hiroyasu, T., Kaneko, M. and Hatanaka, K.: A Parallel Genetic Algorithm with Distributed Environment Scheme, *IEEE International Conference on Systems, Man, and Cybernetics*, Vol.1, pp.695–700 (1999).
6) Ono, I., Mizuguchi, N., Nakashima, N., Ono, N., Nakada, H., Matsuoka, S., Sekiguchi, S. and Tate, S.: Gridifying A Genetic Algorithm for NMR Three-Dimensional Protein Structure Determination by Using Ninf-1 and Ninf-G, *IPSJ Journal*, Vol.46, No.12, pp.369–406 (2005). (in Japanese).
7) Lim, D., Ong, Y.S., Jin, Y., Sendhoff, B. and Lee, B.S.: Efficient Hierarchical Parallel Genetic Algorithms using Grid computing, *Future Generation Computer Systems*, Vol.23, No.4, pp.658–670 (2007).
8) Li, J.M., Wang, X.J., He, R.S. and Chi, Z.X.: An Efficient Fine-grained Parallel Genetic Algorithm Based on GPU-Accelerated, *Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on*, pp.855–862 (online), DOI:10.1109/NPC.2007.108 (2007).
9) Thompson, A.M. and Dunlap, I.B.: Optimization of analytic density functionals by parallel genetic algorithm, *Chemical Physics Letters*, Vol.463, No.1–3, pp.278–282 (2008).
10) Windows Communication Foundation: http://msdn.microsoft.com/en-us/library/dd456779.aspx.
11) Papazoglou, M. and Georgakopoulos, D.: Service-Oriented Computing, *Communications of the ACM*, Vol.46, No.10, pp.25–28 (2003).
12) Zhang, W. and Cheng, G.: A Service-Oriented Distributed Framework-WCF, *Web Information Systems and Mining, International Conference on*, Vol.0, pp.302–305 (2009).
13) Riad, A.M., Hassen, A.E. and Hassen, Q.F.: Design of SOA-based Grid Computing with Enterprise Service Bus, *International Journal on Advances in Information Sciences and Service Sciences*, Vol.2, No.1, pp.71–82 (2010).
14) Kosugi, Y., Oyama, A., Fuji, K. and Kanazaki, M.: Conceptual Design Optimization of Hybrid Rocket Engine, *Proceedings of Space Transportation Symposium* (2009).