International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:4, 2009

Adaptive Distributed Genetic Algorithms and Its VLSI Design

Kazutaka Kobayashi, Norihiko Yoshida, Shuji Narazaki

Abstract—This paper presents a dynamic adaptation scheme for the frequency of inter-deme migration in distributed genetic algorithms (GA), and its VLSI hardware design. Distributed GA, or multi-deme-based GA, uses multiple populations which evolve concurrently. The purpose of dynamic adaptation is to improve convergence performance so as to obtain better solutions. Through simulation experiments, we proved that our scheme achieves better performance than fixed frequency migration schemes.

Keywords—Genetic algorithms, dynamic adaptation, VLSI hardware.

I. Introduction

To obtain drastic performance improvement, it is effective to implement genetic algorithms (GAs) in VLSI hardware. The structure of GA computation and problem representation forms a good basis for VLSI hardware. VLSI hardware implementations of GAs have already been widely studied, some of which concerned specific problems such as pattern matching, scheduling, the traveling salesman problem and image filtering, and some of which are problem-independent.

Their extensions towards parallel GA and distributed GA are also being studied to obtain much better performance improvement. Parallel GA evaluates fitness values of multiple genotypes simultaneously, and thus realizes fine-grained parallel processing. Distributed GA, or multi-deme-based GA, uses multiple populations which are evolving concurrently, and thus realizes coarse-grained parallel processing.

We designed and developed GAP, a general-purpose GA-VLSI with parallel and distributed GA implementations [1]. The distributed GA configuration of GAP is composed of multiple GAPs working concurrently. It is important in distributed GA that some of the genotypes should be “migrated” between demes occasionally in order to prevent isolated evolution and premature convergence. We used the simplest scheme for migration: in every evolution cycle, newly created genotypes are migrated to the next deme.

In this paper, we present a dynamic adaptation scheme for the frequency of inter-deme migration. In short, each processor observes the convergence status of its deme, i. e. observes the gradient of the convergence curve, and determines how often to communicate for migration. The purpose is to reflect convergence properties and adapt dynamically so as to obtain better solutions.

K. Kobayashi is with InterDesign Technologies Inc., Tokyo 105-0014, Japan. N. Yoshida is with Department of Information and Computer Sciences, Saitama University, Saitama 338-8570, Japan (e-mail: yoshida@mail.saitama-u.ac.jp). S. Narazaki is with Department of Computer and Information Sciences, Nagasaki University, Nagasaki 852-8521, Japan.

Manuscript received July 15, 2009; revised September 7, 2009.

We present GAP/D, a multi-deme-based distributed-GA VLSI design, and through simulation experiments, we prove that our scheme achieves better performance than fixed frequency communication schemes without affecting the overall convergence properties.

Section 2 summarizes the design of the original GAP. Section 3 describes its extensions to parallel GA and distributed GA. Section 4 explains dynamic adaptation scheme for intermeme migration, and Section 5 proves its effect by preliminary software-based simulations. Section 6 presents some experiment results, and Section 7 contains some concluding remarks.

II. Design of GAP

Our GA-VLSI system is composed of two modules: a general-purpose problem-independent part for selection, crossover and mutation operations, and a problem-dependent part for fitness evaluation. GAP is the problem-independent module, working together with FEP (Fitness Evaluation Processor) dedicated to problem-dependent fitness evaluation (Fig. 1).

GAP contains some modules for random number generation and population control along with modules for selection, crossover and mutation. FEP must be designed for a given specific problem. GAP and FEP are connected via a population memory. In this system, the below modules are the most innovative. Their details are found elsewhere [1].

Population Memory:

Many other GA-VLSIs have employed the conventional generational GA. In that scheme, the whole population is updated at once when a generation proceeds. Two sets of population memories are required for the current and next generations respectively, and there is overhead for transferring or switching one memory to the other when a generation proceeds.

GAP employs the steady-state GA [2]. In this scheme, the population is updated continuously;

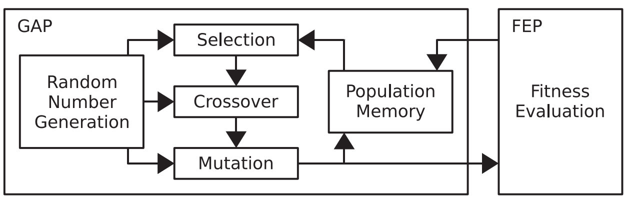

Fig. 1. Basic GAP architecture.

there is actually no concept of generations. Genotypes are passed from GAP to FEP as soon as they are created, and then passed back from FEP to GAP as soon as they are evaluated. Genetic operations and fitness evaluations are overlapped in this way, and the whole system works in a pipeline fashion. Only a single set of population memories is required.

Selection Module:

Most other GA-VLSIs employ the roulette wheel selection scheme for genotype selection. The roulette wheel selection is the most straight-forward scheme: the greater a genotype's fitness is, the more likely that genotype will be selected. However, as can easily be seen, this scheme is a bottleneck for VLSI hardware in respect to both circuit size and performance.

In theoretical research on GA, some alternative selection schemes which are better with regard to convergence properties have been studied and examined [3]. Investigating such theoretical researches, we have introduced a selection scheme which is suitable for VLSI hardware. It is named the "simplified tournament selection", since it is a simplified version of the tournament selection scheme.

Random Number Generation Module:

This module generates a sequence of pseudo-random bit strings using the theory of linear cellular automata (CA). The CA scheme was proved theoretically to generate better random sequences, in the sense that the sequences had a longer cycle length, than the scheme of linear feedback shift registers (LFSR) which has been widely used [4].

From the VLSI implementation standpoint of view, the CA scheme spends more gates than the LFSR scheme. However, the randomness of the random number generation is one of the most crucial in GA implementation, and the CA scheme was proved to be better in this respect. Consequently, we employed the CA scheme.

III. Extensions to Parallel and Distributed GA

As problems to be solved become more complicated, FEPs turn into bottlenecks for the overall GA system performance. Therefore, multiplication of FEPs in a system can be effective for improving performance.

In the research area of theories and software for GA, there have already been many studies on parallel and distributed processing for GA. Parallel GA evaluates fitness values of multiple genotypes simultaneously, and thus realizes fine-grained parallel processing. Distributed GA, or multi-deme-based GA, uses multiple populations which are evolving concurrently, and thus realizes coarse-grained parallel processing.

The basic architecture of GAP hardware design facilitates extensions for parallel GA and distributed GA.

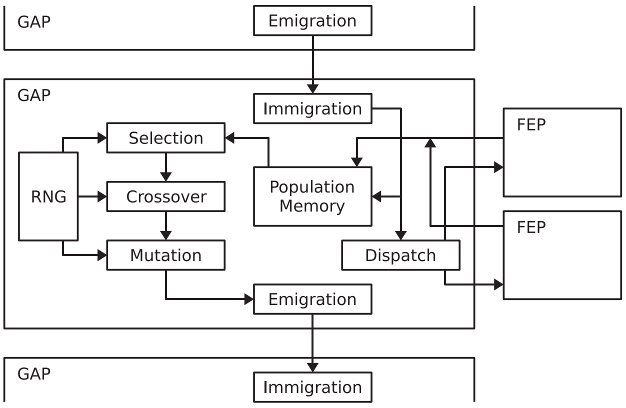

Fig. 2. Parallel and Distributed GAP architecture.

Parallel GA for GAP:

The simplified tournament selection scheme creates two new genotypes and evaluates their fitness values in every evolution cycle. Therefore, two FEP chips can be connected to a GAP, as shown in Fig. 2, and they evaluate the fitness values of the two new genotypes simultaneously. The dispatch module controls these two FEPs. This parallel GA configuration is expected to double the performance of fitness evaluation.

Distributed GA for GAP:

The distributed GA configuration of GAP is composed of multiple GAPs working concurrently. It is important in distributed GA that some of the genotypes should be "migrated" between demes occasionally in order to prevent isolated evolution and premature convergence. At first, we used the simplest scheme for migration: in every evolution cycle, newly created genotypes are migrated to the next deme. GAP chips are connected to each other in a ring form. Newly created genotypes are transferred to the population memory in the next GAP chip via the emigration and immigration modules. This configuration contributes to convergence.

IV. Dynamic Adaptation for Migration Frequency

Isolated evolution in each deme tends to get trapped into a dead-end. This is known as premature convergence, and is prevented by migrating genes occasionally among demes. There are some parameters for inter-deme migration, and "how often to migrate" is the most critical.

Regarding migration frequencies, most distributed GA systems migrate genes in every generation or at randomly chosen intervals. These migration schemes are so simple that they cannot reflect convergence properties, nor adapt dynamically so as to obtain good solutions. A too low frequency leads to premature convergence, whereas a too high frequency spoils the effect of parallel evolution, and system performance due to communication overhead.

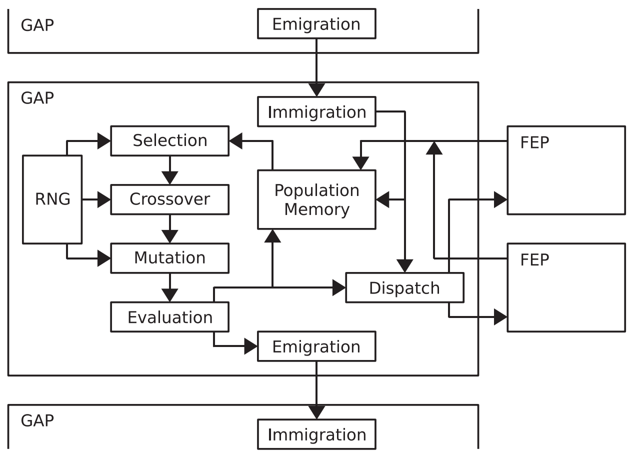

Fig. 3. GAP/D architecture.

The essence of our idea is very simple in its principle. A processor observes the gradient of the convergence curve of its deme, and performs migration when the gradient gets flat, i. e. its deme is about to converge. On each generation t, the processor computes an average fitness value $f(t)$ and its gradient

$$g(t) = \frac{f(t) - f(t - \Delta t)}{\Delta t}$$

It then checks whether $g(t) \leq g_{th}$ or not against a certain pre-defined threshold g_{th}, and performs migration when this condition stands (Fig. 3).

There is a related study for dynamic adaptation based on population distribution [5]. A processor observes standard deviation σ of gene fitness in its demes, and performs migration when σ gets low. However, this scheme disregards absolute values of gene fitness, thus sometimes causes premature convergence towards the low level of fitness. Also, this scheme involves intensive computation, and difficult to fit for hardware implementation.

V. Preliminary Evaluation

To verify the effectiveness of the dynamic adaptation scheme for migration frequencies, we performed some preliminary experiments using software implementation over LAN. For simplicity, it is generational GA, and network communication is broadcast-based.

Fig. 4 shows one result of the experiments, in which each of four lines stands respectively for:

- single: single deme
- 4 cluster: four demes on a single host
- dist, fix: four demes on four hosts, with fixed migration frequency
- dist, var: four demes on four hosts, with adaptive migration frequency

Here, the sample problem is the Rastrigin function minimization, which is from the standard set on benchmark problems for GA system evaluation [6]. It is to find a vector $\vec{x}$ which minimizes

$$f(\vec{x}) = 3.0n + \sum_{i=1}^{n}\{x_i^2 - 3.0cos(2\pi x_i)\}$$

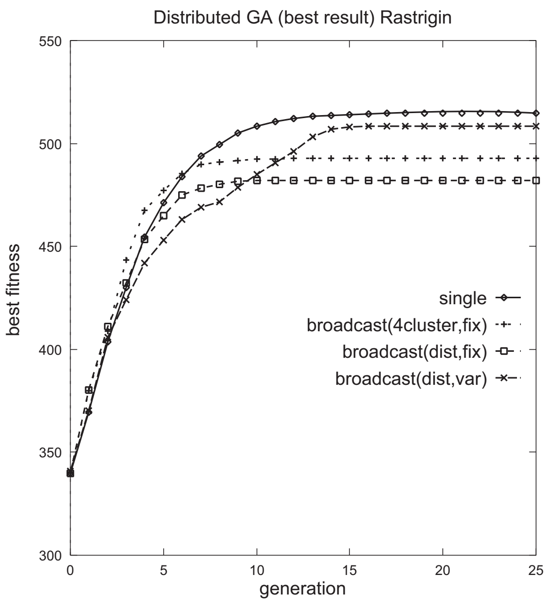

Fig. 4. Software simulation of adaptation.

We observe some bends in the line of GA with adaptive migration frequency around the generations 8 and 13. Comparing the four lines, we also see that the distributed GA with fixed migration frequency suffers from premature convergence, and the adaptive migration frequency scheme improves it up to almost comparable to the single GA.

VI. Simulation Platform and Experiments

We designed the VLSI hardware of GAP using a hardware description language "SFL" [7], assuming the CMOS 0.8 μm process technology. We carried out logic simulation and logic synthesis for preliminary evaluation of the design prior to actual VLSI fabrication. The specification of the prototype is:

Population size:	256
Genotype bit length:	64
Fitness bit length:	24
Crossover probability:	1
Mutation probability:	1/32

The comparisons regarding the number of gates (circuit size) and the number of steps per cycle (speed) between the original GAP and the new GAP/D are as below:

	gates	clocks/cycle
GAP	53890	17
GAP/D	60477	21

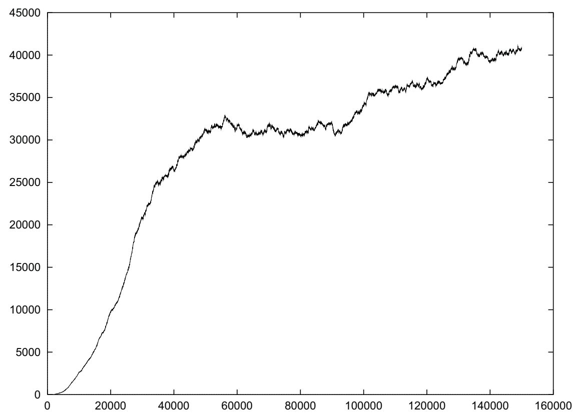

Fig. 5. Convergence on GAP.

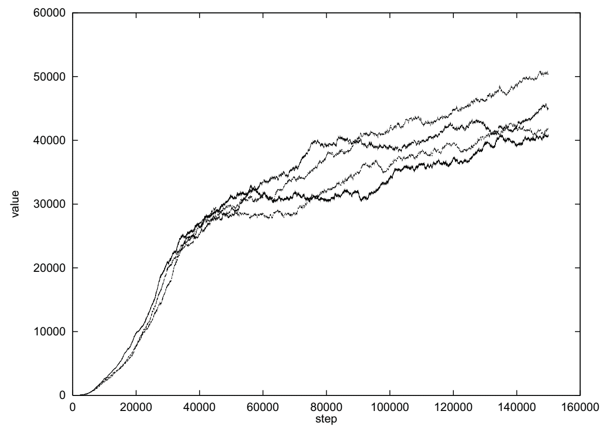

Fig. 6. Convergence on GAP/D.

We did some experiments using a well-known standard example for GA experiments called "Royal-road function" [8], which is simple but slow to converge. This problem is to maximize the below R on the 64bit vector $\vec{x}$ where "$\bigwedge$" is logical "and":

$$R(\vec{x}) = 64D + 32\sum_{k=0}^{1} C_k + 16\sum_{k=0}^{3} B_k + 8\sum_{k=0}^{7} A_k$$

$$A_k = \bigwedge_{i=8k}^{8k+7} x_i, \quad B_k = \bigwedge_{i=16k}^{16k+15} x_i,$$

$$C_k = \bigwedge_{i=32k}^{32k+31} x_i, \quad D = \bigwedge_{i=0}^{63} x_i$$

Fig. 5 shows the convergence of Royal-road function on the original GAP, and Fig. 6 shows the convergence on the new GAP/D, where the threshold is 32, and four curves correspond to four demes respectively. We observed that the former converges to the value of approximately 41,000, while the latter converges to 50,000 at highest. This proves that the adaptive migration scheme improves the quality of the solution result.

VII. Concluding Remarks

We presented a dynamic adaptation scheme for the frequency of inter-deme migration in multi-deme-based distributed GA on GA VLSI. Each processor observes the gradient of the convergence curve, and determines how often to migrate.

Using an example of function minimization problems, we showed that our scheme achieves better performance than fixed frequency migration schemes without affecting the overall convergence properties.

This improvement causes some increases in the circuit size and the execution speed. The increase in the circuit size is not a serious issue. The increase in the execution speed from 17 clocks to 21 clocks per cycle (approximately 124%) can be acceptable, however, we had better improve the technique so as to reduce the increase. It is left for further study.

References

[1] N. Yoshida, T. Yasuoka, T. Moriki, and T. Shimokawa, "VLSI Hardware Design for Genetic Algorithms and Its Parallel and Distributed Extensions", *Int. J. of Knowledge-Based Intelligent Engineering Systems*, Vol. 5, No. 1, 2001, pp. 14–21.

[2] L. Davis (ed.), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

[3] H. Sato, I. Ono and S. Kobayashi, "A New Generation Alternation Model of Genetic Algorithms and Its Assessment" (in Japanese), *J. Japanese Society for Artificial Intelligence*, Vol. 12, No. 5, 1997, pp. 734-744.

[4] M. Serra, T. Slater, J. C. Muzi and D. M. Miller, "The Analysis of One-Dimensional Linear Cellular Automata and Their Aliasing Properties", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 9, No. 7, 1990, pp. 767-788.

[5] M. Munetomo, Y. Takai, and Y. Sato, "An Efficient Migration Scheme for Subpopulation-based Asynchronously Parallel Genetic Algorithms", *Proc. 5th Int. Conf. on GA*, 1993, p. 649.

[6] M. A. Potter and K. A. De Jong, "A Cooperative Coevolutionary Approach to Function Optimization", *Proc. Third Conf. on Parallel Problem Solving From Nature*, 1994, pp. 249–257.

[7] Y. Nakamura, K. Oguri, et al., "High-Level Synthesis Design at NTT Systems Labs", *IEICE Trans. on Inf. & Syst.*, Vol. E76-D, No.9, 1993, pp. 1047–1054.

[8] M. Mitchell, and S. Forrest, "Fitness Landscapes: Royal Road Functions", *Handbook of Evolutionary Computation* (T. Back, D. Fogel, and Z. Michalewicz, eds.), Oxford, 1997.