AN LSI IMPLEMENTATION OF AN ADAPTIVE GENETIC ALGORITHM WITH ON-THE-FLY CROSSOVER OPERATOR SELECTION

Shin'ichi Wakabayashi *Tetsushi Koide* *Naoyoshi Toshine*
Mutsuaki Goto *Yoshikatsu Nakayama* *Koichi Hatta*

Faculty of Engineering, Hiroshima University
4 -1 Kagamiyama 1 chome, Higashi-Hiroshima 739-8527 JAPAN
Tel.: +81-824-24-7678 Fax.: +81-824-22-7195 E-mail: wakaba@computer.org

ABSTRACT

This paper describes an LSI implementation of a genetic algorithm (GA), called the Genetic Algorithm Accelerator (GAA) chip. The GAA chip is an LSI implementation of a GA, in which two types of crossover operators are supported, and the operator to be actually used in the algorithm is not fixed in advance, but dynamically selected for each pair of chromosomes in the algorithm execution. The GAA chip has been designed with the Verilog HDL and simulated with some benchmark functions. According to the simulation, the GAA chip will run with 50MHz clock in maximum. The chip has been fabricated with the CMOS 0.5 μm standard cell technology.

1. INTRODUCTION

Genetic algorithms (*GAs*) [3] are known as one of robust heuristic algorithms for complex optimization problems in various fields of engineering. GA provides robust capability of exploring in the solution space of a given problem. There are two notorious problems on GAs to realize their performance. One is the parameter tuning. Since a GA has many parameters including types of genetic operators, it is difficult to set parameters to appropriate values so as to draw out a maximum capability of GA. The other problem of GAs is the computation time. We often encounter the case that a GA requires large amount of computation time, and due to the limitation of computation time, a GA produces poor solutions.

To solve the former problem, adaptive GAs have been investigated [1]. An adaptive GA is the GA, for which some parameters including the types of genetic operators are not fixed in advance, but are determined during the execution of the algorithm. The authors have also proposed an adaptive GA, which selects a crossover operator in an adaptive manner during the execution of the algorithm, since the crossover has in general a large effect on the performance of a GA [4]. In the proposed adaptive strategy, an appropriate crossover operator among two types of crossover operators are automatically selected for each pair of chromosomes (individuals) to be crossed over. To realize this mechanism, a new measure called the *elite degree* has been proposed. The elite degree was devised to show the potential proficiency of an individual in the particular generation.

To reduce the execution time of GA, there are some ideas including parallel and/or distributed processing of GAs and hardware implementation of GAs [5, 6, 7, 9, 10]. As hardware implementation of a GA, Scott *et al.* proposed a hardware-based genetic algorithm (HGA), which was implemented on a set of field-programmable gate arrays (FPGAs) [7]. The HGA is based on the standard Simple Genetic Algorithm (SGA) [3]. Sano *et al.* proposed a SIMD GA-machine, which was designed to implement on FPGAs [6]. The basic algorithm of [6] is also the SGA, which was modified to be executed on a SIMD parallel computer. Yoshida *et al.* also proposed a hardware-based GA, called the Genetic Algorithm Processor (GAP) [10]. The GAP is based on a steady-state GA, which does not have the generation.

This paper presents an LSI implementation of an adaptive genetic algorithm, called the Genetic Algorithm Accelerator (GAA) chip. The GAA is a hardware implementation of an adaptive genetic algorithm, which we have proposed in [4]. The GAA has two types of crossover operators, that is, two-point crossover and uniform crossover, and it dynamically selects one of them to apply it to a pair of chromosomes (individuals) to be crossed according to the status of chromosomes. In [4], we have shown that the adaptive GA outperformed both the SGA and adaptive GAs with other crossover selection strategies. We modified this original adaptive GA so that it is easy to implement it as hardware.

This paper is organized as follows. Section 2 will give an overview of GAs and the adaptive strategy of crossover selection adopted in the GAA. Section 3 will present the architecture of the GAA. Section 4 will describe the LSI implementation of the GAA, and finally Section 5 will give some conclusion.

2. AN ADAPTIVE GENETIC ALGORITHM

A genetic algorithm (GA) is a general framework of heuristics for numerical/combinatorial optimization problems. A GA is based on natural selection and evolution process of life, and is known as a robust, powerful, and problem-independent optimization technique. In the normal GA, the crossover operator to be used in the algorithm is determined in advance, and is fixed during the algorithm execution. However, we often encounter the case that more than one crossover operators could be used to the given problem, and the characteristics of those operators are different. For example, there are many problems for which both of two-point and uniform crossover operators could be applied. However, the properties of those operators are different. The former has a nondestructive nature of potential good schemata of a chromosome, whereas the latter has a high ability to explore the solution space [2]. So, to fix the type of crossover operator in advance might loose the potential capability of GAs.

To resolve the problem above mentioned, we have proposed an adaptive strategy of crossover selection [4]. The following is the mechanism of on-the-fly selection of crossover operators, which is adopted in the GAA.

Assume that the problem to be solved is a maximization problem. Let $P^T = \{x_1^T, x_2^T, \cdots, x_n^T\}$ be a set of chromosomes of the generation T, where x_i^T is an i-th chromosome in the population. Let $f(x_i^T)$ be the value of fitness of chromosome x_i^T, and *ave_f*(P^T) and *max_f*(P^T) be the average and the maximum values of fitness values of all chromosomes in the population P^T.

Now, we define chromosome $x_i^T \in P^T$ as an *elite* if the following condition holds.

$$f(x_i^T) \quad \geq \quad \textit{ave\_f}(P^T) +$$

$$\alpha \times (max\_f(P^T) - ave\_f(P^T)) \qquad (1)$$

where α is the *elite decision factor*, and in the current design of GAA, α can be set to 0.25 or 0.5.

Next, we define the *elite degree* of chromosome x_i^T. Let *Anc*(x_i^T, j), $(j \geq 1)$, be a set of ancestors of chromosome x_i^T in the generation $T - j$. Let *Elite*$(x_i^T, j) \subseteq$ *Anc*(x_i^T, j) be a subset of *Anc*(x_i^T, j) such that each chromosome in *Elite*(x_i^T, j) is an elite in the generation $T - j$. Then, the elite degree of x_i^T, denoted $E\_deg(x_i^T)$, is defined as follows.

$$E\_deg(x_i^T) = \frac{\sum_{j=1}^{lmax} \{|Elite(x_i^T, j)| \times \beta^j\}}{\sum_{j=1}^{lmax} \{|Anc(x_i^T, j)| \times \beta^j\}} \qquad (2)$$

$\beta (0 < \beta \leq 1)$ is the *elite influence factor*, and *lmax* is the range of elite degree calculation. In the current design of GAA, *lmax* is set to 4, and for each chromosome data, the number of elite chromosomes of previous generations, i.e., $|Elite(x_i^T, j)|$, $1 \leq j \leq 4$, are kept in the memory. The whole calculation of expression (2) is realized with the table look-up. Note that, as *lmax* = 4, 14 bits are required to be used as address bits of this table look-up. The contents of the table is precomputed and stored by the external CPU.

Crossover selection based on the elite degree will be done in the GAA as follows. Let

$$E1(x_i^T) = \begin{cases} 1 & \text{if } x_i^T \text{ is an elite.} \\ 0 & \text{otherwise.} \end{cases}$$

Assume that x_i^T and x_j^T are selected to be crossed over. Then, let

$$E\_deg2(x_i^T, x_j^T) = E\_deg(x_i^T) + E\_deg(x_j^T) + E1(x_i^T) + E1(x_j^T) \qquad (3)$$

If $E\_deg2(x_i^T, x_j^T) \geq T_{cross}$, then a two-point crossover operator will be applied since the parents are expected to have good schemata, and two-point crossover is less destructive than uniform crossover. If $E\_deg2(x_i^T, x_j^T) < T_{cross}$, then a uniform crossover operator will be applied to explore the solution space more widely to find better chromosomes (solutions). We call T_{cross} the *elite threshold*. In the GAA, T_{cross} is a user-defined parameter.

Except the crossover selection, which was explained above, the GAA implements the generation-based standard GA with the roulette wheel selection and elitist strategy.

3. THE GAA

3.1. Overview

The Genetic Algorithm Accelerator (GAA) is an LSI implementation of a GA with adaptive selection of crossover operators explained in Section 2. To make the GAA general-purpose, evaluation of fitness of each chromosome is not implemented in the GAA chip. Since the fitness function depends on the problem to be solved, we assume that fitness evaluation will be done outside the GAA chip, and it might be implemented on reprogrammable FPGAs or a microprocessor.

The GAA chip can be used as a co-processor of the main CPU of a personal computer (PC) or workstation. The GAA chip can be connected to the local bus of a PC with a small amount of interface logic. All computations except the fitness calculation can be done by the GAA chip. The fitness calculation may be performed with the CPU of the PC. Since the CPU only needs to calculate the fitness of each chromosome, the total computation time will be much reduced.

Table 1. Specifications of the GAA chip.

Population size†	64, 128
Crossover†	2-point, uniform, adaptive
Generation†	512, 1024, 2048, 4096
Selection	roulette selection + elitist strategy
Crossover rate†	$0/256 \sim 255/256$
Mutation rate†	$0/4096 \sim 255/4096$
Elite threshold†	$0.0 \sim 4.0$
Elite decision factor†	0.25, 0.5
Elite influence factor†	table look-up
♯bits of an individual	64 bits
Fitness	16 bits

The GAA chip can be also used as a hardware-based GA System. The GAA chip with the reprogrammable hardware devices such as FPGAs is implemented as one system. FPGAs are used to calculate the fitness function of the given problem. The logic of FPGAs are described with a hardware description language such as Verilog, which will be synthesized, and loaded from the host machine. Since all computations will be done by hardware, computation time can be drastically reduced.

3.2. Specifications of the GAA

Specifications of the GAA chip are summarized in Table 1. In the table, the items with † show the user programmable parameters.

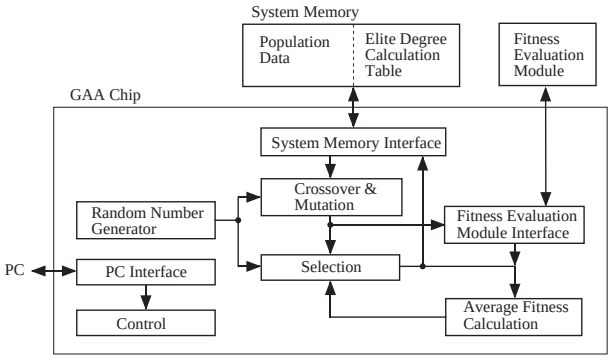

Figure 1. Overall design of the GAA.

3.3. Overall Design

Figure 1 shows the overall design of the GAA. The GAA consists of three modules as follows.

(1) GAA chip.

The GAA chip is the main module of the whole system, and performs all operations except the fitness calculation. The GAA chip itself consists of several submodules (units), which will be explained in detail in the next subsection.

(2) System Memory (SM).

The 32Kword static RAM is attached to the GAA chip, where 1 word = 16 bits. This memory is used to keep the information of each chromosome. Information of each chromosome consists of (a) code (64bits), (b) fitness value (16bits), and (c) family tree information (15bits). The family tree information of a chromosome is used to calculate the elite degree. The system memory is also used as the data table for the calculation of the elite degree. Initial values of the system memory are set by the external CPU.

(3) Fitness Evaluation Module (FEM).

Fitness of each individual will be evaluated by this module. The FEM may be implemented on FPGAs or a CPU. Each time a pair of chromosomes are created by the Crossover and Mutation

Unit in the GAA chip, data of chromosomes are passed to the FEM through the FEM Interface. Since the FEM receives two chromosome data, parallel processing of fitness evaluation may be achieved if the FEM is designed to calculate the values of fitness function for two data in parallel.

3.4. The GAA Chip

The GAA chip consists of the following submodules (units).

(1) Crossover and Mutation Unit (CMU).

This unit is used to mate two chromosomes with specified crossover operators. The resultant chromosomes thus produced will be modified with the mutation operators. Let *p_cross* and *p_mutate* be the crossover and mutation rates given by the user, and *pop* represent the population size. Then, $(pop \times p\_cross)/2$ pairs of chromosomes are randomly selected and mated with the user specified crossover operators. If the adaptive crossover is specified, the GAA calculates the elite degree of each chromosome by accessing the System Memory to read the precomputed value of the elite degree.

This unit runs in parallel with the external Fitness Evaluation Module (FEM). Two newly created chromosomes after mating are passed to the FEM. After passing the chromosome data, the CMU continues its function. Every 149 clock cycles, the CMU will passes the chromosome data to the external FEM. So, if the fitness calculation in the FEM can be done in less than 149 clock cycles, the execution of the FEM is completely overlapped with the execution of CMU, and no clock cycles dedicated to the fitness evaluation is required.

(2) Selection Unit (SU).

The roulette wheel selection and elitist strategy are realized in this unit. The roulette wheel selection is the procedure to reproduce the chromosomes of a new generation. The number of chromosomes reproduced from one chromosome depends on its fitness value, that is, the expected number of copies of a chromosome x_i^T is represented by $f(x_i^T)/ave\_f(P^T)$. Since the division requires a large amount of hardware resources, in the SU, we implement the roulette wheel selection with subtraction and comparison operations.

The GAA adopts the elitist strategy, i.e., we always keep the best chromosome ever produced in the previous generations.

(3) Average Fitness Calculation Unit (AFCU).

The values of $ave\_f(P^T)$ and $max\_f(P^T)$ are calculated in each generation.

(4) Random Number Generator (RNG).

Pseudorandom numbers are generated by using a cellular automaton based algorithm [8]. A random number is a 24 bit integer. The initial value can be set by the user.

(5) System Memory Interface (SMI).

The System Memory Interface provides the interface function between the GAA chip and the System Memory. The data width is 16 bits, and the address width is 15 bits.

(6) Fitness Evaluation Module Interface (FEMI).

This unit provides the interface function between the GAA chip and the Fitness Evaluation Module. The interface is based on handshaking, and any design of the FEM can be attached to the GAA chip through the FEMI.

(7) PC Interface (PCI).

This unit provides the interface function between the GAA chip and the PC. The PC can control the behavior of the GAA. For example, if the PC wants to start the GAA, the PC simply sends the GO signal. When the GAA finishes, then the DONE signal will be sent back to the PC.

(8) Control Unit (CU).

This unit controls the overall behavior of the GAA chip.

The whole circuit was designed as a single-phase clock synchronized circuit.

4. LSI IMPLEMENTATION

4.1. Chip Design

The GAA chip has been designed with the Verilog HDL, and synthesized with the Synopsis Design Compiler. The layout design has been done with the Cadence Cell Ensemble. The chip has been fabricated as a $4.8mm^2$ standard cell chip with $0.5\mu m$ CMOS technology with two metal layers. Table 2 shows the synthesis result of the chip. A cell in this table means a basic gate such as AND, OR, etc., a compound gate such as AND-OR-INV, and a latch such as D-FF. The number of pins only shows the number of signal pins of the chip. The post layout simulation showed that the chip will be able to run with a 50 MHz clock. The GAA chip was realized as a 120 pin PGA. Figure 2 shows the chip image of the GAA chip.

Table 2. Synthesis result of the GAA chip.

Number of pins	76
Number of nets	5915
Number of cells	5890
Clock frequency	50*MHz*

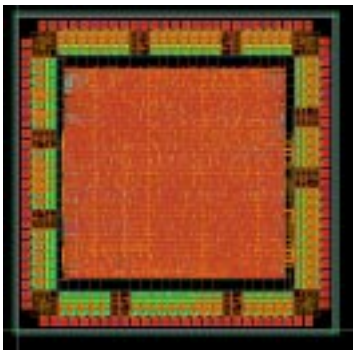

Figure 2. Chip image of the GAA chip.

Table 3. GAA v.s. GENESIS 5.0.

crossover	best/gen.	GAA	GENESIS
2-point	best	0.0	0.0
	gen.	161	207
uniform	best	0.0	0.0
	gen.	86	184
adaptive	best	0.0	0.0
	gen.	82	46

4.2. Evaluation

To evaluate the GAA chip, we performed several simulation experiments. First, we compared the GAA with a general-purpose GA software, called GENESIS, which is a well-known free software of GA package. We compared the GAA with GENESIS by solving several benchmark test functions. Among results obtained from the experiment, due to the limitation of space, we only show the results for DeJong's test function $f3$ in Table 3. The value of optimal solution is 0.0. In the simulation, we executed the GAA and GENESIS in 10 times, and compared the

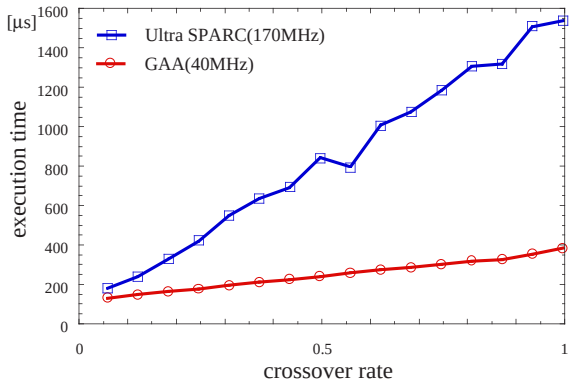

Figure 3. Execution time v.s. crossover rate.

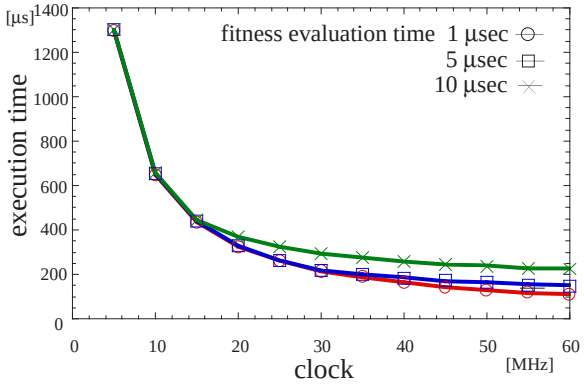

Figure 4. Execution time v.s. clock frequency.

best results in those 10 runs as well as the average number of generations, in which the best results were found. As a crossover operator, we used three types of crossover operators including the proposed adaptive one. Two other operators were 2-point and uniform crossover operators. In the table, "best" shows the best result in 10 runs. "gen." shows the average number of generations to obtain best results.

From Table 3, we found that, for any case, the GAA obtained optimal solutions. For both GAA and GENESIS, when the proposed adaptive crossover operator was used, the final best solutions were found in a smallest number of generations compared with the cases with other crossover operators. From the viewpoint of ability as an optimization procedure, there is no significant difference between GAA and GENESIS. From experiments with other test functions, the same conclusion was obtained.

Next, we show the relationship between the execution time and the crossover rate. Execution time of the GAA as well as a software GA depends on the crossover rate. In this experiment, we made a software version of the GAA, whose function is the same as the GAA, and compared it with the GAA chip. Figure 3 shows the experimental results. From this graph, we see that the execution time depends on the crossover rate. Furthermore, the GAA chip with 40MHz clock is 2 to 4 times faster than a 170 MHz UltraSPARC processor.

The final simulation results show the relationship between the execution time and clock frequency. Figure 4 shows the execution time per one generation. Note that the execution time depends on both the clock frequency and the fitness evaluation time of the external fitness evaluation module. In this graph, we consider the three cases. In the first case, the fitness evaluation of one pair of chromosomes is assumed to require 1 microsecond. In the second and third cases, 5 and 10 microseconds are needed. From this graph, if the clock frequency is greater than 15 MHz, the execution time also depends on the fitness evaluation time.

We also made an experimental board, on which the GAA system was constructed, and connected with a PC. A test program was also developed to control the GAA system as well as to serve as an external device to compute the fitness function. We observed that the chip was successfully working when the clock frequency was 10 MHz. In the current design of the board, SRAMs with the access time of 85 nanoseconds are used as System Memory, that determines the clock frequency of the board. So, if we use faster SRAMs, the board may work with 20 MHz or higher clock.

5. CONCLUSION

This paper presented an LSI implementation of an adaptive GA. The chip was fabricated with 4.8 mm^2 CMOS standard cell technology. Test results showed that all the functions of the chip was successfully implemented.

As future work, there are several extensions of this work. First, more adaptive mechanisms for parameter tuning can be incorporated into the chip. We have already proposed an adaptive selection method of the mutation rate of GAs. Introducing the adaptive tuning of more parameters may improve the qualities of solutions as well as reduce the computation time. Second, more parallel processing can be introduced to the chip. For example, to reduce the computation time, crossover and mutation can be run in parallel for more than one pairs of chromosomes.

ACKNOWLEDGMENT

The VLSI chip in this study has been fabricated in the chip fabrication program of VLSI Design and Education Center(VDEC), the University of Tokyo with the collaboration by NTT Electronics Corporation and Dai Nippon Printing Corporation. This research is supported in part by Grant-in-Aid for Scientific Research (C) (No. 10680356) from the Ministry of Education, Science, Sports and Culture of Japan.

REFERENCES

[1] L. Davis: "Adapting operator probabilities in genetic algorithms," *Proc. 3rd International Conference on Genetic Algorithms*, pp.61–69 (1989).

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

[3] D. E. Goldberg: *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing Co. (1989).

[4] K. Hatta, K. Matsuda, S. Wakabayashi, and T. Koide: "On-the-fly crossover adaptation of genetic algorithms," *Proc. IEE/IEEE Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp.197–202 (1997).

[5] D. Patrick, P. Green and T. York: "A distributed genetic algorithm environment for UNIX workstation clusters," *Proc. IEE/IEEE Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp.69–74 (1997).

[6] M. Sano, T. Inoue and Y. Takahashi: "A design of the SIMD GA-machine with FPGA," *IPSJ SIG Notes*, 97-ARC-125-6 (1997), in Japanese.

[7] S. D. Scott, A. Samal and S. Seth: "HGA: A hardware-based genetic algorithm," *Proc. ACM/SIGDA 3rd International Symposium on FPGA*, pp.53–59 (1995).

[8] 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. CAD*, 9, 7, pp.767–788 (1990).

[9] J. Stender (eds.): *Parallel Genetic Algorithms: Theory and Applications*, IOS Press (1993).

[10] N. Yoshida, T. Moriki and T. Yasuoka: "Design of genetic algorithm VLSI using SFL," *10th PARTHENON Workshop*, pp.63–70 (1997), in Japanese.