Fast Implementation of Genetic Algorithm Based on Software/Hardware Co-design Method

Wenqian Zhou*
School of Microelectronics, Hefei University of Technology, Hefei, China

*Corresponding author: wq.zhou@mail.hfut.edu.cn

Abstract. Genetic algorithm (GA) has been widely used since it was proposed. However, due to the complexity of the algorithm itself, the fact that the actual problems have huge data and scale, and that it needs to meet certain real-time requirements, the acceleration of genetic algorithm has become a hot spot in both academia and industry field. Previous work have been done on the acceleration of GA, which primarily focuses on the software aspect, while the existing work in hardware aspect lacks adaptability for different industrial scenarios. Therefore, this paper puts forward a novel architecture based on software/hardware co-design method, in which the software produces random numbers, and the hardware, taking full account of the characteristics of sequential circuit, is implemented in a hierarchical pipeline architecture. This architecture accelerates the algorithm while providing certain flexibility. This paper completes the simulation on MATLAB and Vivado, and the total running time is 987.4μs. Compared with the pure software implementation, the speed is increased by 73.9 times.

Keywords: genetic algorithm, software/hardware co-design, pipeline.

1. Introduction
In both theory research and practical applications, there are a large number of problems related to optimization. Since the establishment of Bionics in the mid-1950s, people have been actively seeking inspiration from natural organisms and have developed a series of simulated evolutionary algorithms based on natural selection and evolutionary mechanism, which are suitable for the optimization of complex problems in the real world [1]. The genetic algorithm, first proposed by Professor John H. Holland of the University of Michigan and his students in the early 1960s is one of the algorithms, which is based on Darwin’s theory of evolution [2, 3]. Compared with traditional optimization methods, bionics based optimization method is essentially a probabilistic parallel search method. Individual behaviors in the group are independent and interrelated, showing a high degree of parallel, random and adaptive [4].

The application research of GA is more abundant than the theoretical research and has penetrated into many disciplines. In the field of automation, Michelewicz [5] used floating-point coded GA to study the discrete-time optimal control problem, Wang Yongding [6, 7] used GA to optimize the blade of hydraulic turbine, and Yang Kun [8] used GA to optimize the layout of traditional camshaft; In the field of economics and management, Feng Weidong [9] used GA to study the selection of [10, 11] used GA to study the focusing method of wavefront shaping, and Chu Xiaoli [12] used GA to select the...
wavelength. Genetic algorithm is also applied to computational mathematics [13], electronics [14],
electric power [15], aerospace [16] and other disciplines.

With the continuous development of science and technology, the scale of the problem continues to
expand. In the face of the increasingly complex search space in practical issues, the standard genetic
algorithm (also known as simple genetic algorithm, SGA) takes too long because of its complex
operation process. Therefore, previous work have been done to optimize the algorithm. However, the
research on the algorithm is more focused on the software aspect, and the main platform is based on
CPU so far. Srinivas [17] and others proposed adaptive genetic algorithm (AGA), which enables
individuals to adjust crossover and mutation probability according to different fitness, and effectively
improves the convergence speed of the algorithm. Lu Peng [18] and others proposed a parallel genetic
algorithm based on hash function acceleration, which can reduce the repeated calculation of individual
fitness, and thus increase the running speed by more than three times. Liu Xiaoping [19] parallelelized the
basic genetic algorithm by using the master-slave model, aiming at the problem of a large amount
of calculation of fitness evaluation, and achieved a nearly linear speedup. With the continuous improvement
of speed requirements, people begin to accelerate the algorithm from hardware aspect. Some people
rewrite the algorithm in a hardware-friendly way. For example, Wang Zhidan [20] proposes a hardware
implementation structure, which adopts the method of serial design, uses random League selection,
double-crossing, and bit mutation, and it takes 113.8us; The hardware architecture proposed by Wu
Chunying [21] adopts the design method of multi-population parallel and partial pipeline in a single
population. It uses competitive selection, sequential crossover, exchange and mutation, and it takes
3300us; The hardware implementation structure proposed by Wang yuti [22] adopts a multi-population
parallel, single subpopulation serial design method, which uses random League selection, two-point
crossover, multi-point mutation, and it takes 295us. These implementation methods greatly improve the
execution speed of genetic algorithm.

The essence of genetic algorithm is an efficient parallel search method for solving problems [23],
which determines that the algorithm is more suitable for parallel computing. Therefore, the
computing efficiency is limited by the serial computer system on which the softwares are based, leading
to the computing speed being too low to meet the real-time requirements in practical applications.
However, the algorithm implemented by pure hardware also has its inadequacy of less flexible. To be
specific, many parameters, such as random numbers, are fixed values once the work is done. The attempt
to modify them requires recompiling, which consumes a lot of time. In order to resolve the issues, a
novel architecture based on the software/hardware co-design method is designed. Considering the
complexity of the operation, parallelism and resource consumption, etc, random number generation and
coding of the algorithm are implemented by software, and the rest operations are implemented by
hardware, which not only improves the execution speed of the algorithm, but also expands its
application scope.

2. Method

2.1. The operation process of genetic algorithm
As shown in Fig.1, the general steps of genetic algorithm are:
1) Encode:
   According to different usage scenarios, choose the corresponding coding methods. The most
   common ones are binary coding [24] and floating point coding [25, 26].
2) Generate initialization population:
   An initial population of N individuals was randomly generated and the chromosome length was
determined.
3) Fitness calculation:
   The fitness of each chromosome in the population was calculated.
4) Judge whether the ending conditions are met:
The ending condition can be that the evolution algebra of cosine setting has been completed, or the optimal individual has been reached, or the optimal individual has not been improved for several consecutive generations. If the condition is met, the algorithm is ended and the optimal individual is output; If not, repeat steps 5, 6 and 7.

5) Select:
According to the principle that the higher the fitness, the greater the probability of selection, two individuals are selected from the population as the father and mother.

6) Crossover:
Chromosomes of both parents are extracted and crossed by probability $P_c$ to produce offspring.

7) Mutation:
Chromosomes of offspring were mutated with probability $P_m$.
End loop

2.2. Implementation using software/hardware co-design method
In this paper, a top-down, software/hardware collaborative design method is adopted. Considering the complexity of the operation, parallelism and resource consumption, etc, random number generation and encoding are implemented by software, and the rest of the operations are implemented by hardware. The hardware part is divided into six modules: selection module, crossover module, mutation module, fitness module, control module and storage module. The hardware structure is shown in Fig.2.

3. Experiment
The computer version used in this paper is: CORE(TM) i7 2.80GHz, software simulation tool: MATLAB R2016a, hardware simulation tool: Vivado 2020.1.
3.1. Select module

The most basic and commonly used selection method is Roulette Wheel [24]. Its basic idea is that the probability of each individual being selected in the population is proportional to its fitness. However, considering the parallelism of the operation, we choose a more hardware-friendly Tournament Selection Model [20], that is, S individuals are chosen from the population every time to compare the size of fitness, and the one with the highest fitness is inherited into the next generation population, then this process is repeated until the number of individuals in the next generation reaches the preset number. In this paper, S is selected as 2.

In order to complete the design pattern of the pipeline, two input individuals must be selected within one cycle and transferred to the next module, which is the cross module. Considering that the crossover module needs two individuals to operate, this paper uses spatial parallelism to achieve acceleration.

The ports of the select module are shown in Fig.3. In the input port, clk and rst are clock signal and reset signal, in1 and in2 are individuals read from the storage module, fit1 and fit2 are corresponding fitness values read from the storage module, and S-enable is the module enable signal sent by the control module. In the output port, out is the selected individual, and enable_C is the enable signal transmitted to the crossover module.

According to Fig.4, the select module starts to work at the rising edge of clk after S-enable is pulled high, compares and outputs the input with larger fit value, pulls high enable_C for the next module.

---

**Figure 2.** Implementation structure of hardware

**Figure 3.** Port diagram of selection module
3.2. Cross module

Crossover operation, analogous to chromosome crossover in biology, refers to the replacement and recombination of partial structures of two-parent individuals to form two new individuals. In this paper, we choose the single point crossover method, that is, randomly set a crossover point in the individual coding string, and then exchange partial chromosomes of two parent individuals at this point so as to obtain new offspring chromosomes.

As the second stage of the pipeline, the crossover module receives individuals from the selection module and random numbers ran1 and ran2 from ROM. The 8-bit ran1 is compared with the preset 8-bit crossover probability $P_c$. If it is larger, crossover operation is neglected, and direct output is performed, otherwise the crossover operation is performed. Complete the above operations in one cycle and send the output to the mutation module.

The 4-bit ran2 determines the crossing position, and $P_c$ selects 0.908, representing as $8'b1110_0111$.

```
in1[15:0]   out1[15:0]
in2[15:0]   out2[15:0]
ran1[7:0]    
ran2[3:0]    
C_enable    enable_M
clk          
rst          
```

**Figure 5.** Port diagram of cross module

The ports of the crossover module are shown in Fig.5. In the input port, clk and rst are clock signal and reset signal, in1 and in2 are individuals read from two selection modules, ran1 and ran2 are random numbers, and C_enable is the module enable signal sent by the selection module. In the output port, out1 and out2 are the output individuals after the crossover operation, and enable_M is the enable signal passed to the mutation module.

The Cross module starts working at the rising edge of clk after the C_enable signal is pulled high. If the value of ran1 is smaller than $P_c$, the crossover module will crossover the individuals. Otherwise, individuals maintain the same value. And the enable signal enable_M of the latter module is pulled high.
3.3. Mutation module
Mutation operation is carried out on an individual. In binary-coded individuals, some values in the string are reversed, that is, 0 is used to replace 1, and vice versa, thus a new individual is generated. The purpose of introducing a mutation into genetic algorithm is to [21]:
(1) Improve the local random search ability of the algorithm.
(2) Introduce new genes to prevent premature convergence.
In this paper, the basic single-point mutation is adopted, and the probability $P_m$ is 0.098, representing as 8’b0001 0110. The module reads random numbers ran3 and ran4 from ROM. Ran3 is used for comparison. If it is larger than $P_m$, it will be output without mutation. Otherwise, it will be mutated, and the position of mutation will be determined by ran4.

![Figure 7. Port diagram of mutation module](image)

The ports of the mutation module are shown in Fig.7. In the input port, clk and rst are clock signal and reset signal, in1 and in2 are individuals read from the cross module, ran3 and ran4 are random numbers, and M_enable is the enable signal of the module, which is sent by the cross module. In the output port, out1 and out2 are the output individuals after the mutation operation and enable_F is the enable signal passed to the fitness module.

As can be seen from Fig.8, the mutation module starts to work at the rising edge of clk after M_enable is pulled high: enable F is pulled high, whether it will be mutated is determined according to the comparison between ran4 and constant, the result is output, and the enable signal enable_F of fitness module is pulled high.

3.4. Fitness module
The fitness module performs the calculation operation of individual fitness in GA. It is the most complex module in the whole system, so internal optimization is required. In this paper, a test function of

![Figure 6. Simulation waveform diagram of cross module](image)
DeJong [27] is used, namely

\[ f(x) = \sum_{i=1}^{k \leq 2} x_i^2, 0 \leq x_i \leq 255 \]  

(1)

This paper uses a 8-bit pipelined multiplier for calculation because of the 8-bit multiplication. Therefore, there is a four-stage pipeline in the fitness module: three stages for the multiplier and one for the adder. In this way, the subdivision within the module effectively improves the maximum operating clock frequency of the whole system.

The ports of the mutation module are shown in Fig.9. In1 and In2 receive data from ROM in the initial population stage (S1), and receive data from the mutation module in the subsequent stage (S2). Out1 and Out2 output the received original data. Fit1 and Fit2 are the calculated fitness. Maxfit and maxout are obtained by comparing the values of In1 and In2 with the values of maxfit and maxout in the previous cycle. Addr is the address of the storage module, and the value is obtained by the internal counter of the module.

It can be seen from Fig.10 that the value of fitness is calculated by pipeline multiplier and adder, and is delayed for four cycles. The maximum values are judged and output in two cycles. In order to correspond the address of the individual and the fitness in the storage module, add the address of the individual with four.

3.5. Control module

The FSM of the control module is shown in Fig.11 (b).

IDEL: Idle state after initialization.

S1: Initial population stage. During this state, the enable signal of the fitness module is given by the control module, and the fitness module receives the random individuals pre-stored in the ROM as in1 and in2. In this paper, the number of individuals in one generation is set to
Figure 10. Simulation waveform diagram of fitness module

![Simulation waveform diagram of fitness module](image)

Figure 11. (a) Port diagram of control module (b) FSM of control module

![Port diagram of control module](image)

![FSM of control module](image)

32, and the number of generations is set to 64. Cnt_i_max is 16 because the initial population is provided by two ROMs.

S2: Normal working stage. During this state, the enable signal of the fitness module is given by the mutation module, and the fitness module receives out1 and out2 of the mutation module as inputs in1 and in2. The number of generations is set to 64, therefore cnt_gen_max is 64. In this state, there is also a counter cnt_i_n, which records the number of individuals in the current generation. Individuals are provided by two dual-port RAMs, that is, four individuals are provided at a time, thus cnt_i_n_max is 8. When cnt_i_n reaches cnt_i_n_max, cnt_i_n is cleared and cnt_gen is increased by one.

FINISH: The end state. The system enters FINISH state when cnt_gen reaches cnt_gen_max, and maintains the state until the reset signal arrives.

As can be seen from Fig.12 (a), when the initial generation is full, the system enters S2 at the rising edge of the next clock, and the counter in S2 starts counting. When the number of individuals in the contemporary generation is full, the generation counter is increased by one, and when the number of generations is full, it enters FINISH state.

As can be seen from Fig.12 (b), on the rising edge of the next clock after the generation number is full, the write enable is pulled low, the storage module and the selection module are turned off.

3.6. Storage module

The storage module consists of four true dual-port RAMs. Individuals and fitness values are stored separately, in which RAM0 and RAM1 store individuals, and RAM2 and RAM3 store fitness values.
Figure 12. Simulation waveform diagram of control module

Port A of all four RAMs works normally, while Port B is read only. And the address of port B is the address of port A increased by one.

According to the above analysis, it takes four cycles to get the result of the fitness values of RAM2 and RAM3. Therefore, in order to correspond to the address of the individual and fitness value, the address of RAM0 and RAM1 is increased by four.

3.7. Overall simulation results

The simulation result of the top module is shown in Fig.13.

With the proposed architecture, the running time of the software is 977\(\mu s\), and the running time of hardware is 10.4\(\mu s\), a total of 987.4\(\mu s\). Simulation with pure software shows that the running time is 73\(ms\), as shown in Table 1.

4. Conclusion

First and foremost, this paper achieves a profound understanding of GA in its operation characteristics, and proposes a novel architecture using both hardware and software. Secondly, this paper uses the pipeline structure of selection, crossover and mutation modules, and selects

| Test Function |
|---------------|
| \(f(x)\)     |
| \(f(x)\)     |

| Implementation Method | Running Time |
|-----------------------|--------------|
| pure software         | 73 (ms)      |
| software/hardware co-design | 987.4 (\(\mu s\)) |
hardware-friendly genetic operators. Thirdly, taking the operating characteristics of sequential circuits into full consideration, the complex fitness module is subdivided into pipeline structure, so the module with long delay does not drag down the overall maximum operating clock frequency.

Simulation results show that the proposed architecture has excellent performance compared to that of pure software, increasing the running speed by 73.9 times, and the design goal is achieved.

However, the GA algorithm used in this paper is the most basic one, and it does not aim at any specific practical application. In the future design, we can:

Use the adaptive algorithm to change the crossover and mutation probability flexibly to further accelerate the convergence speed of the algorithm.

Combine the algorithm with specific applications. Given the practical problems, the same idea of this paper can be adopted, which means to further increase the clock frequency by rediving the complex modules with pipeline structure.

References
[1] Xi Yugeng and Chai Tianyou. A survey of genetic algorithms. 13(6):697–708, 1996.
[2] John H Holland. Concerning efficient adaptive systems.
[3] Self-Organizing Systems, 230, 1962.
[4] John Henry Holland et al. Adaptation in natural and artificial systems: an introductory analysis with applica- tions to biology, control, and artificial intelligence. MIT press, 1992.
[5] Chen Zhe and Geng Guosheng. Bionic optimization algorithms and their applications in signal processing. In Journal of Data Acquisition and Processing, pages 662–681, 2018.
[6] Zbigniew Michalewicz, Jack B Krawczyk, Mohammad Kazemi, and Cezary Z Janikow. Genetic algorithms and optimal control problems. In 29th IEEE conference on decision and control, pages 1664–1666. IEEE, 1990.
[7] Wang Yongding and Li Ningye. Optimization and re- search of tidal turbine blades based on genetic algorithm. In Journal of Mechanical Strength, pages 327–332, 2021.
[8] Zbigniew Michalewicz, Cezary Z Janikow, and Jacek B Krawczyk. A modified genetic algorithm for optimal control problems. Computers & Mathematics with Ap- plications, 23(12):83–94, 1992.
[9] Yang Kun and Zou Jian. Optimization and simulation for facility layout of camshaft production line based on genetic algorithm. Machine Tool Hydraulics, 49(1):124–129, 2021.
[10] Feng Weidong and Chen Jian. Partner selection process and optimization model of dynamic alliance based on genetic algorithm. JTsinghua Univ(SciTech), 40(10):120–124, 2000.
[11] Tang Xin. Research on the rapid optical focusing of laser after the wavefront shaping based on the genetic algorithm penetrating scattering medium, 2018.
[12] Feng Qi. Research on Multi-point Light Focusing of Feedback-based Wavefront Shaping Technology With Ge-netic Algorithm. PhD thesis, 2020.
[13] Chu Xiaoli and Yuan Hongfu. Progress and application of spectral data pretreatment and wavelength selection methods in nir analytical technique. Progress in Chem- istry, 14(4).
[14] Feng-Tse Lin, Cheng-Yan Kao, and Ching-Chi Hsu. Applying the genetic approach to simulated annealing in solving some np-hard problems. IEEE Transactions on systems, man, and cybernetics, 23(6):1752–1767, 1993.
[15] Klaus Glaßmacher, Axel Heß, and Gerhard Zimmermann. A genetic algorithm for global improvement of macrocell layouts. In ICCD, pages 306–313, 1991.
[16] Cui Ting and SunYuanzhang. Reactive power optimiza- tion of power system based on improved niche genetic algorithm. Proceedings of the CSEE, 31(19):43–50, 2011.
[17] ChenGenshe and Chen Xinhai. Design of automatic rendezvous controller using genetic algorithm. Journal of Northwestern Polytechnical University, 12(2):247–253, 1994.
[18] M. Srinivas and L. M. Patnaik. Adaptive probabilistic- ties of crossover and mutation in genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics, 24(4):656–667, 1994.
[20] Lu Peng and Xiao Xiaoqiang. Parallel genetic algorithm acceleration method based on hash function. Journal of Computer Applications, pages 124–127, 2020.

[21] Liu Xiaoping and An Zhulin. Master-slave parallel genetic algorithm framework on mpi. Journal of System Simulation, 40(S1):1938–1956, 2004.

[22] Wang Zhidan. Research on the realization technology of genetic algorithms based on fpga, 2006.

[23] Wu Chunying. Research on hardware technology of genetic algorithms based on fpga, 2004.

[24] Wang Yuti. Research on hardware implementation technique of parallel genetic algorithms based on fpga, 2008.

[25] Chen Guoliang. Genetic algorithm and its application. The Posts and Telecommunications Press, 1996.

[26] David E Golberg. Genetic algorithms in search, optimization, and machine learning. Addison Wesley, 1989(102):36, 1989.

[27] David E Goldberg et al. Real-coded genetic algorithms, virtual alphabets and blocking. Citeseer, 1990.

[28] Alden H Wright. Genetic algorithms for real parameter optimization. In Foundations of genetic algorithms, volume 1, pages 205–218. Elsevier, 1991.

[29] KA De Jong. An analysis of the behavior of a class of genetic adaptive systems [ph. d. thesis]. University of Michigan, 1975.