A PIPELINED BASED FPGA IMPLEMENTATION OF A GENETIC ALGORITHM

Nonel Thirer
HIT – Holon Institute of Technology, 54 Golomb Street, Holon, Israel

Keywords: Genetic Algorithm, Pipeline, Resources, Subpopulations, Flexible Implementation.

Abstract: Many problems common to the electrical and electronics field can be solved by finding a target function and its minimum or maximum. For such problems, usually an analytical solution is not implementable, and therefore iterative algorithms are used. One such efficient algorithm is the Genetic Algorithm (GA). The GA imitates the biological evolution process, finding the solution by implementing the "natural selection" principle, which asserts that the strong has higher chances to survive. The GA is an iterative procedure which operates on a population of individuals called "chromosomes" or "possible solutions" (usually represented by a binary code) and performs several processes on the population individuals, in order to produce a new population - the same as in the biological evolution. Using the algorithm on large populations requires substantial hardware resources. Also, naturally, the amount of time necessary to reach a solution increases, due to the greater number of iterations needed. In this paper, we present an FPGA pipelined based method designed to implement a GA, which provides a high-speed solution for large populations, with a minimum of resources. This outcome is obtained by a procedure which operates sequentially with parts of the population. In addition, an immigration unit is defined to provide an efficient communication between these parts in different iterations. Moreover, some possible solutions to improve our method are analyzed.

1 INTRODUCTION

The principal steps of the GA (Affenzeller, 2009) are:

Population initialization – build a random initial bank of chromosomes.

Fitness calculation – calculate the "Fitness score" for each chromosome in the current chromosome bank. The Fitness score of a chromosome is the amount of adjustments needed to solve the main problem. If a chromosome that solves the main problem is found, the GA is stopped.

Selection – the GA chooses several chromosomes from the current bank, with respect to their calculated fitness score.

Crossover – two or more of the chosen chromosomes randomly switch their bits and build up a new chromosome.

Mutation – one random bit is toggled in the newly built chromosome.

The stages are repeated until a new chromosome bank (population) is built and the process continues with the new population.

The software implementations of the algorithm convert these steps to a multi phase process. The FPGA (Field Programmable Gate Array) implementations provide special hardware blocks for every phase and also general blocks, which are common to many phases (Mao, 1999). Evidently, this implementation on large populations requires substantial hardware resources. In addition, the time needed until arriving to a solution increases, due to a greater number of iterations. The use of a parallel architecture improves the speed but requires many resources (Tatsuhiro, 2006).

Problem–specific and also problem–independent designs and implementations are available (Tiago, 2004).

A good algorithm must provide a high speed solution in parallel to resources' optimization for a problem - independent FPGA implementation.

2 A NOVEL IMPLEMENTATION ALGORITHM

We proposed a novel implementation algorithm based on a pipelined system working with parts ("subpopulations") of the entire populations. These

Thirer N..
A PIPELINED BASED FPGA IMPLEMENTATION OF A GENETIC ALGORITHM.
DOI: 10.5220/0003687703430345
In *Proceedings of the International Conference on Evolutionary Computation Theory and Applications* (ECTA-2011), pages 343-345
ISBN: 978-989-8425-83-6

subpopulations are not isolated and the best individuals are transferred between subpopulations.

The initial population is divided to n subpopulations and a GA multi stage process is complete for each subpopulation by an m phases pipeline procedure.

To process all these subpopulations, instead of using a parallel configuration with n pipelines (Tatsuhiro, 2006) we propose a sequential procedure by using this m phases pipeline. Thus, only resources for a single pipeline are necessary.

Evidently, for best results, the initial population must be divided to n subpopulations, where n = m.

By using a single pipeline, the working time is longer than using parallel pipelines, but the sequential operation with the subpopulations increases the throughput.

A "transfer" unit is introduced, to permit an efficient communication between the subpopulations in different iterations and also to improve the fitness score of each subpopulation. In our method, the transfer is provided by added to each subpopulation the best p members of a previously evaluated subpopulation. This, except in the case of the first two subpopulations in the initialization phase (in this case p members of another subpopulations must be added randomly).

In each procedure, the Fitness block will provide the best p candidates to be added to the next subpopulation procedure.

2.1 Implementation of a Four Stages GA

In the four stages GA, the pipeline contains four special blocks:

M - Working memory (including the transfer unit), E - Evaluation, S - Selection, CM - Crossover and Mutation.

P0 is our initialization population. We divide it to four subpopulations: P0_1, P0_2, P0_3, P0_4.

P0_11, P0_21, P0_31, P0_41 are the new subpopulations after the first iteration,

P0_12, P0_22, P0_32, P0_42 are the new subpopulations after the second iteration and P0_1x, P0_2x, P0_3x, P0_4x are the new subpopulations after the x-th iteration.

In this case, the four stages pipeline procedure will work in the following mode:

Time	Pipeline Blocks			
	M	E	S	CM
t1	P0_1			
t2	P0_2	P0_1 best to M		
t3	P0_3 +best P0_1	P0_2 best to M	P0_1	
t4	P0_4 +best P0_2	P0_3 best to M	P0_2	P0_1
t5	P0_11 +best P0_3	P0_4 best to M	P0_3	P0_2
t6	P0_21 +best P0_4	P0_11 best to M	P0_4	P0_3
t7	P0_31 +best P0_11	P0_21 best to M	P0_11	P0_4
t8	P0_41 +best P0_21	P0_31 best to M	P0_21	P0_11

And so forth ad so on.

By example, the system status at time t7 is as follows:

The Memory Block M contains P0_31 which is the first generation of the third subpopulation (P0_3 which already passed all the pipeline stages) and the best individuals of the previous evaluated P0_11.

The Evaluation Block E evaluates the P0_21 subpopulation (fitness calculations) to provide data for the Transfer unit and also for the Selection Block. Obviously, if a chromosome that solves the main problem is found, the process is stopped.

The Selection Block S chooses the chromosomes of the P0_11 subpopulation to be the next parents.

The Crossover and Mutation Block CM provides a new generation (P0_41) of the P0_4 subpopulation.

In this way at time t5 the entire population is evaluated and a new generation is obtained at time t4 to t7.

2.2 General Blocks

The implementation of the GA requires also general blocks used by two or more special blocks. Thus, a major component of the hardware architecture of the GA is the source of pseudo randomly noise. Our pseudo randomly binary number generator (PRBG), based on LCA (Linear Cell Automata) is used to select the parents groups (from the fitness output) and also (in parallel) to provide the bits for the crossover and mutation phase (Godkin, 2010).

The PRGB is based on 2q+1 flip-flops and their outputs (0 to 2q) provide a 2q+1 pseudo random bit sequence (PRBS). The GA uses 2q sequences: bit 0 up to bit 2q, bit one up to bit 2q and bit 0, and so on.

Those 2q+1 bits sequences are then divided into

smaller sequences, of q bit each, using the same principle, bit 0 up to bit q-1, bit 1 up to bit q, and so on.

In this manner, the PRGB provides the GA with many random numbers (values: 0 to 2^k - 1) at each iteration.

2.3 Implementation Parameters

The hardware implementation of the algorithm on any specific FPGA device requires the user to define some hardware parameters, such as:

- the size of the population members (n)
- the number of bits of each population individual
- the size of the immigration population - the number of the best members (p)
- the size of the PRNG (2k+1)
- the maximum number of generations (gn)
- the precision used in fitness estimation
- the number of bits of the crossover
- the number of bits and the probability of the mutation

Careful and precise definitions of the above mentioned parameters will provide us with a flexible implementation. The above hardware parameters depend, of course, on the nature of the problem needing a solution, and are restricted by the specific FPGA chip characteristics.

2.4 Extended Solution

The presented algorithm, based on a pipelined system working with "subpopulations" of the entire populations, can be adapted to any s-stages Genetic Algorithm (working, for example, with memory and transfer as separate phases and also with crossover and mutation as separate phases) by dividing the initial population to s subpopulations.

The flip-flop array of the PRNG could easily be expanded if the amount of random numbers supplied to GA is not enough to make all the necessary calculations during a single iteration. Also, the PRNG component can be defined and implemented by using other algorithms, for example LFSR - linear feedback shift registers (Nedjah, 2007). Another option is using a predefined component.

As it is usually the case with genetic algorithm implementations, there are no guarantees that using these methods will provide a better solution in the next generation. A simple comparator unit can be added to detect and store the best solution found to a specific point in time, and after gn generations the user may use this best solution, if a chromosome that solves the main problem wasn't found.

3 CONCLUSIONS

The above portrayed method provides a flexible and compact implementation of a given problem, using a genetic algorithm and an FPGA device.

The method allows us to work with a large size population by using a small amount of resources. It does so by dividing the initial population into "subpopulations". The "transfer" step provides the necessary interconnection between the members of the entire population.

The presented pipeline organization permits an implementation of a four stages algorithm. For a five or more stages algorithm the pipeline must be reconsidered and adapted to usage as a five or more stages pipeline.

As it is often the case with genetic algorithm implementations, the runtime can't be exactly calculated. However, it is certain that the time needed to accomplish a single iteration using a pipeline procedure is longer that without pipeline, but a considerable increase of the throughput will be obtained.

To provide a flexible implementation, some hardware parameters should be defined prior to the FPGA implementation. Also, external defined components (as PRBG) can be used.

REFERENCES

M. Affenzeller A. O., 2009. Genetic Algorithms and Genetic Programming – Modern Concepts and Practical Applications, CRC Press, USA.

N. Nedjah, L. M. Mourelle, 2007. An efficient problem-independent hardware implementation of genetic algorithms., *Neurocomputing* 71, p.88-94.

Mao F. So, Angus Wu, 1999. FPGA Implementation of Four –Step Genetic Search Algorithm., *Electronics, Circuits and Systems, Proc. of ICECS '99*, vol.2 p.11 43-1146.

Tatshuito Tachibana A. O., 2006. Flexible Implementation of Genetic Algorithms on FPGA, *Proc. of the ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays*, FPGA, USA, February 22-24, 2006, 9 pages

Tiago Carvalho Oliveira, Valfredo Pilla Jr, 2004. An Implementation of Compact Genetic Algorithm on FPGA for extrinsic evolvable Hardware, *IEEE Transactions on Evolutionary Computation*, p.1143-1146.

Godkin Andrey, Nonel Thirer, 2010. A FPGA Implementation of Hardware Based Accelerator for a Genetic Algorithm, *Proc. of IEEE 26-th Conv. of Electrical and Electronics Engineers in Israel*, p.578-580.