Selected aspects and tradeoffs in transistor level implementation of genetic algorithms

Sławomir Jeżewski and Rafał Długosz,

Abstract— **In this paper we focus on the issues of hardware implementation of genetic algorithms (GA) in hardware. In their classic implementation, the genetic algorithms search for a global minimum or maximum of a multidimensional function called the fitness function. If the problem, i.e. the fitness function, is too complex for a brute force search, we can look for a solution based on GA. In this situation we obtain desired results by performing parallel calculations on the "generation" of candidate solutions randomly distributed over the input data space. For these candidates we evaluate the fitness function and then we breed a next generation. During breeding we use operators that mimic chromosomal crossing to exchange of features between the candidates, and mutation operations to introduce new features into the population. The literature research shows that more than sixty different crossing algorithms are used with the GA in different purposes. Such a large number of crossing algorithms is a serious problem when developing a hardware solution. In this paper, we are reviewing a deployment of selected crossing operators in specialized hardware.**

I. Introduction

Genetic algorithms (GA) as a mathematical concept were born in the 1960s in the works of J. Holland [1]. At the turn of the seventies and eighties, it is noticed the first significant increase in their popularity, when classic papers of De Jong [2], Goldberg [3] and Davis [4] were published. These works caused the canonical rise of the genetic algorithms and have identified the potential scope of application of them. Since then GAs have enjoyed uninterrupted, constant interest.

The scientific world is systematically informed about effective applications of GAs in various optimization tasks both in science and in technology. The second stream of publication presents alternations of well-known algorithms invented on the base of a particular real word problem. As of today, we know more than sixty crossing algorithms for binary data and eighty for data encoded with real numbers [5]. Similarly, we can enumerate twenty seven binary mutation algorithms and forty seven for real numbers [5]. These lists are not exhaustive. There is a large amount of publications on the attempts to implement GAs in the form of the field programmable gate arrays (FPGAs) [6], [7], [8], [14], [13], [12], as well as in graphics processing unit (GPU) [11], [15]. These examples are related to the industrial use of the GAs, for example, in automotive technologies and in Artificial Intelligence in general.

S. Jeżewski is with the Institute of Informatics, College of Social and Media Culture, ul Św Józefa 23/35, 87-100 Toruń, Poland, E-mail: slawomir.jezewski@wsksim.edu.pl

R. Długosz is with the UTP University of Science and Technology, Faculty of Telecommunication, Computer Science and Electrical Engineering, ul. Kaliskiego 7, 85-796, Bydgoszcz, Poland, and with Delphi Poland, ul. Podgórki Tynieckie 2, 30-399, Kraków, Poland, E-mail: rafal.dlugosz@gmail.com

A typical GA is based on the calculation scheme shown below:

```
Data: $n$ dimensional space $[R, C, I]^n$ defined as $K^n$,
      fitness function $F^n \in K^n$, population P: set of
      points $X \in K^n$
Result: sample with highest value of fitness function
/* generate start population                              */
for $X^n \subseteq P$ do
    $X^N \leftarrow random(K^n)$ /* random coordinates    */
end
/* search fitness space                                   */
repeat
    $R \leftarrow F(P)$ /* evaluate fitness function      */
    $S \leftarrow Select(R, P)$ /* Select samples         */
    $S_1 \leftarrow CrosoverOperator(S)$ /* perform crossover
      */
    $P \leftarrow MutationOperator(S_1)$ /* perform mutation
      */
until is_optimal(P);
```

Algorithm 1: Genetic algorithm calculation scheme

where $[R, C, I]^n$, denotes an n-dimensional space with real, integer or binary coordinates.

In these algorithms, the following blocks play the key role: the fitness function, the candidate selection, the mutation and the crossing functions. The first two functions are run for each population element (sample), while the remaining functions are run for selected ones. The crossing operator is run on a pair of samples or more with a probability of close to unity (0.8 – 0.9) and either exchanges a subset of features between samples or determines the arithmetic mean for continuous features. In contrast, the mutation operator is run with little probability of values not exceeding 0.05 and introduces random changes to selected features. When implementing such algorithms, the researcher selects the fitness function, the appropriate crossing algorithm, determines the probability of crossing, then selects the mutation algorithm and their probability. The execution time of this algorithm depends on the number of samples in the population, the number of the fea-

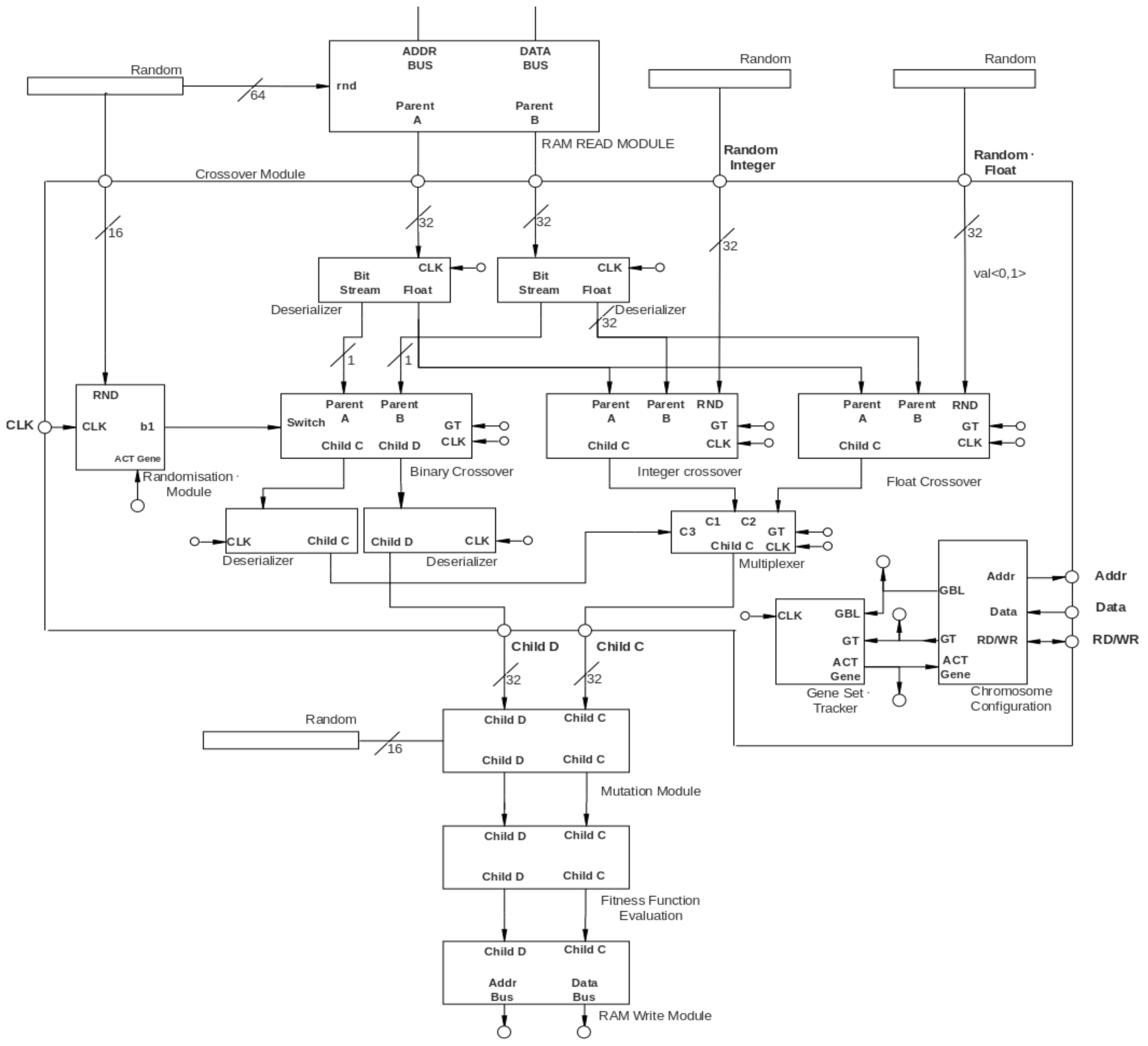

Fig. 1. Functional Block Diagram of Crossover Algorithm

tures (the size of the data space, n) and the complexity of selected functions. Because increasing the number of samples positively affects the probability of detecting the global minimum / maximum, the algorithm are usually started with large files. The above statements introduce the main requirements for the structure to be designed, i.e. how to choose the crossing and the mutation algorithms, and how to develop the structure with massive parallel computation abilities.

II. Implementation specific requirements for genetic algorithms

The promising area for the use of genetic algorithms is in searching for a motion path of the mobile robot [16], [17], [19], [18], [20]. A similar problem, however much more challenging, will exist in case of autonomous driving. The planning algorithm is expected to operate in real time, to have 3D navigation capabilities, bypassing mobile and stationary obstacles and optimizing the energy consumed by the system. In this case searching for a path is a NP-complete problem, which further additional non-geometric constrains, so the computation time increases quickly with the number of nodes.

To timely obtain optimal solutions an efficient implementation of the operators i.e crossing and mutation function is a must. For this reason in this paper we focus on it. The structure of the operator functions strongly depends on the real problem being solved by the GA. In software the operator function may be flexibly modeled in any way. In hardware the situation is more challenging, as the structure of the function cannot be changed. For this reason it is necessary to look for programmable solutions on one hand, and simple on another. There is a trade-off between these criteria.

III. Proposed implementation of the crossing algorithm in FPGA

Successful implementation of GE algorithm in hardware (ASIC - application specific integrated circuit or FPGA) should present at least following features:

- Population size should be programmable, scaled up to $> 10^6$ samples,
- Parent and child population should be stored in external memory SRAM/DRAM to avoid limitations,
- Chromosome composition should be configurable, e.g. number of binary, integer, and floating point genes,
- Crossover algorithms should be selectable via configuration registers, separately for binary and integer and floating point modules,
- Crossover algorithm should be as paralleled as possible to obtain full speed of hardware.

These constrains lead us to an idea of a stream based crossover algorithm, which is configured via a set of registers.

The algorithm Data Flow (shown in Fig. 1) starts with the Data Read Module (DRM) responsible for selecting parents and reading them from the DRAM/SRAM memory. Then parents' chromosomes are feed into the two gene de-serializers, which are entry to crossover module. The bit stream is feed into binary crossover sub-module, the 32 bit variables go into an integer or float crossing sub-modules depending of the type. Each of the sub-modules is enabled by Gene Type signal. The binary crossover sub-module accepts parent chromosomes as a bit stream and inverts A→C , B→D data streams according to the "switch" signal which is generated by the randomization (RM) module.

Floating and integer point crossover modules at the moment support three kind of crossing algorithms chosen by the configuration register, respectively:

$$C_i = \frac{A_i + B_i}{2} \tag{1}$$

$$C_i = \alpha A_i + (1 - \alpha) B_i, \tag{2}$$

$$C_i = \alpha (B_i - A_i) + A_i \tag{3}$$

They are at the moment the most expensive part of design in terms of LUT's and CLB' slices. Data produced by crossing sub-modules are feed into the multiplexer and then to the Mutation module and subsequently to Data Write Module (DWM).

Inherent part of the design is a chromosome configuration. It is formed by a four 32-bit registers (Fig. 2). Each register defines 4 gene blocks (geneset) containing up to 2^6 genes each. The lower 6 bits contains the

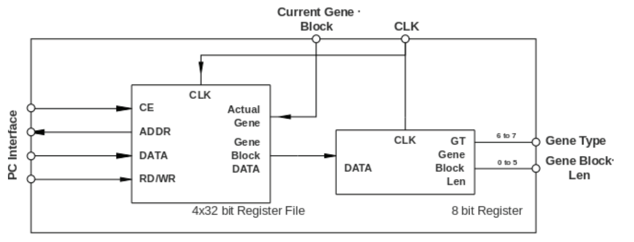

Fig. 2. Functional Diagram of the Chromosome Configuration Block

number of genes, while the upper two contain the gene type.

Supported geneset types are binary genes (00), integer genes (01), floating genes (10), terminator gene (11). The termination gene informs module that chromosome evaluation is finished and the next pair of parents should be selected for processing.

In addition to the main data flow there is a Gene Set Tracker module consisting of three counters and generating to the rest of the system: the number of current gene (CG), the current gene set (CGS) and its type (GT). The GST module is shown in Fig. 1 It receives input from the main clock and chromosome configuration registers. The GST, Serializers and Deserializers, as well as the Gene Configuration are the backbone of the crossing algorithm and at the same time they are the part of the design that is independent of the selected crossing algorithm either binary or floating point. It is one of the advantage of our proposed solution. A flexibility of the the design is also a plus. We are able to solve problems with up to $2^{10} - 1$ genes of mixed types with chromosome length up to 2^{12} bytes, which is no limitation in practical problems. A practical limitation lies elsewhere. Our design can handle only bi-parent crossing algorithms. This is a well-thought-out choice, because the speed optimization of the multi-parent core has other key parameters.

IV. Randomization module

A variety of binary crossing algorithms is hidden mainly in the **R**andomization **M**odule. This module is responsible for creating a switching signal for the **B**inary **C**rossover **M**odule (BCM) and cross signal for the **I**nteger/**F**loating **C**rossover **M**odule (ICM / FCM) (See Fig. 1). In practice, RM is responsible for drawing k random numbers that form the intersection points in the chromosome, where the k parameter is dependent on the type of the crossing algorithm. In our implementation this parameter it being set through the configuration registers. The RM block also generates a 1-bit random number that is needed in the BCM, ICM and FCM sub-modules. This 1-bit random variable is a hardware realization of statement 4 encountered in numerous crossover algorithms [5].

TABLE I

Resource usage from the simulation/synthesis tool

Resource type	Used	Fixed	Available
Slice LUTs	2560	0	41000
LUT as Logic	744	0	41000
LUT as Memory	66	0	13400
Slice Registers	1808	0	82000

$$if(\alpha > 0.5) \tag{4}$$

A random 16-bit variable stored in the shift register can be treated as a source of random bits. As long as the bits state '1' has probability equal to 0.5, it is an equivalent of the equation (4). Three random bits with integer comparison is a sufficient approximation of any $\alpha > \beta$ comparison, where β has two significant digits.

V. Conclusions

In this paper, we proposed a functional concept of a hardware module that implements the genetic algorithm of bi-parent genetic crossing. The module construction is based on a configurable chromosome structure that can handle binary, integer and float genes simultaneously. The large number of genes in the 2^{10} chromosome causes the available RAM to be the only practical limitation. The presented module can work with populations of arbitrary/configurable sizes up to millions of samples. The module is optimized for speed. The investigations show that for problems encoded with binary genes, the algorithm achieves the highest crossing rate of 1 gene per clock tick. For floating point calculations, the rate is slower and results from the time of floating point multiplication. In case where FCM calculations limit the throughput of the solution is necessary to instantiate two or more CM modules in parallel.

Our design follows requirements presented in section III, which is cornerstone of the hardware implementation. The structure, in which the RM block is distinct from BCM and FCM, greatly simplifies solution and further development. With the structure we support most of algorithms described in [5] including: 1-point **C**rossover **A**lgorithm k-point CA, Shuffle CA, Reduced Surogate CA, Uniform CA, Discrete Crossover CA, Flat CA. Some statistics concerning our design are shown in Table I. The presented FPGA implementation is a prototype of a target ASIC realization.

References

[1] J.H. Holland "Outline for bilogical theory of adaptive systems", *Jurnal of the ACM*, vol. 3, no. 3, 1963, pp. 297-314

[2] K.A. De Jong "An analysis of the behaviour of a class of genetic adaptation systemsOutline for bilogical theory of adaptive systems", *doctoral dissertation* University of Michigan , 1975

[3] D.E Goldberg "Genetic algorithms in search, optimisation, and machine learning", *Addison Wesley, Reading*, 1989

[4] L.Davis (ed) "Handbook of Genetic Algorithms", *Van Nostrand Reinhold*, New York

[5] T.D. Gwiazda "Algorytmy Genetyczne kompendium", *Wydawnictwo Naukowe PWN*, Warszawa 2007

[6] M.S Ben Ameur, A. Sakly, A. Mtibaa, , "Implementation of Genetic Algorithms Using FPGA Technology", *Proceedings of the Annual FPGA Conference* (FPGAworld'12), vol. 71, issue. 1-3, pp. 3:1-3:9, Dec. 2012.

[7] N .Nedjahadia L. de Macedo Mourelle, "An efficient problem-independent hardware implementation of genetic algorithms", *Neurocomputing*, vol. 71, issue. 1-3, pp. 88-94, Dec. 2007.

[8] W. Tang, L. Yip, "Hardware Implementation of Genetic Algorithms Using FPGA", *47th IEEE International Midwest Symposium on Circuits and Systems* (MWSCAS'04), Germany, 2004, Volume I

[9] Levy, Erez and David, Omid E. and Netanyahu, Nathan S. "Genetic algorithms and deep learning for automatic painter classification", *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation* (GECCO'14), VAncouver BC Canada, 2014, pp.1143-1150,

[10] Franco, María A. and Krasnogor, Natalio and Bacardit, Jaume "Speeding Up the Evaluation of Evolutionary Learning Systems Using GPGPUs", *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation* (GECCO'10), Portland, Oregon, USA, 2010, pp.1039-1046,

[11] Rojas-Galeano, Sergio and Rodriguez, Nestor "A Memory Efficient and Continuous-valued Compact EDA for Large Scale Problems", *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation* (GECCO'12), Philadelphia, Pennsylvania, USA, 2012, pp.281-288,

[12] Krömer, Pavel and Platoš, Jan "Genetic Algorithm for Sampling from Scale-free Data and Networks", *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation* (GECCO'14), Vancouver, BC, Canada, 2014, pp.793-800,

[13] Vernekar, D. B. Vernekar, G. Malhotra, V. Colaco, "Reconfigurable FPGA Using Genetic Algorithm", *Proceedings of the International Conference and Workshop on Emerging Trends in Technology* (ICWET'10), Mumbai, Maharashtra, India, 2010, pp.493-497,

[14] L. Guo, A. I. Funie, D. B. Thomas, H. Fu, W. Luk "Parallel Genetic Algorithms on Multiple FPGAs", *SIGARCH Comput. Archit. News* (HEART'15), Mumbai, Maharashtra, India, vol 43, Issue 4, Sept. 2016, pp.86-93,

[15] Ma, Yunfeng and Indrusiak, Leandro Soares, "Hardware-Accelerated Parallel Genetic Algorithm for Fitness Functions with Variable Execution Times", *Proceedings of the Genetic and Evolutionary Computation Conference 2016* (GECCO'16), Denver, Colorado, USA, 2016, pp.829-836,

[16] Sung Gil Park ,Shingo Mabu, Kotaro Hirasawa "Robust Genetic Network Programming using SARSA Learning for autonomous robots", *ICROS-SICE International Joint Conference 2009* (ICCAS-SICE'09), Fukuoka, JAPAN, 2009, pp.523-527,

[17] H. Katagiri and K. Hirasama and J. Hu "Genetic network programming - application to intelligent agents", *Systems, Man, and Cybernetics, 2000 IEEE International Conference on* (ICSMC'09), vol 5, 2000, pp.3829-3834,

[18] M. A. Bauda and C. Bazot and S. Larnier "Real-time ground marking analysis for safe trajectories of autonomous mobile robots", *2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics* (ECMSM'17), 2017, pp.1-6,

[19] T. C. E. Cheng and C. T. Ng and E. Levner and B. Kriheli "A Fast Algorithm for Detecting Hidden Objects by Smart Mobile Robots", *2017 IEEE International Conference on Smart Computing (SMARTCOMP)* (SMARTCOMP'17), Hong Kong, China, 2017, pp.1-6,

[20] T. R. Schäfle and S. Mohamed and N. Uchiyama and O. Sawodny "Coverage path planning for mobile robots using genetic algorithm with energy optimization", *2016 International Electronics Symposium (IES)* (IES'16), 2016, pp.99-104,