Applied Soft Computing xxx (2017) xxx–xxx

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A fully customizable hardware implementation for general purpose genetic algorithms

Murat Peker

Department of Electrical and Electronics Engineering, Nigde Omer Halisdemir University, Nigde 51240, Turkey

ARTICLE INFO

Article history:
Received 10 May 2017
Received in revised form
16 September 2017
Accepted 25 September 2017
Available online xxx

Keywords:
Field programmable gate arrays
Genetic algorithms
Hardware design languages

ABSTRACT

In this work, a fully customizable general purpose genetic algorithm (GA) IP core has been proposed for field programmable gate arrays (FPGAs) using the pipeline and parallel architectures to speed up the GA process. The proposed system is implemented on FPGA and coded with very high speed integrated circuits (VHSIC) hardware description language (VHDL). The GA operators and the fitness functions are designed in a modular structure to enable the use of these modules asynchronously. The VHDL code is written with generic parameters to allow the customization of almost every parameter of the proposed FPGA IP Core depending on the problem. The proposed architecture synthesized and tested on Altera DE2-115 board with approximately 12% logic elements utilization. Results are obtained from standard optimization benchmark functions and the traveling salesman problem (TSP). In the hardware experiments, the proposed FPGA IP Core has been found the global optimum solutions for all of the standard benchmark functions and TSP. The clock cycle per generation value of the proposed FPGA IP Core has been decreased up to approximately 95% when compared with the existing implementations. For the TSP case, the proposed FPGA IP Core has reduced the run-time of the compared work approximately 75% and with optimized parameters, the reducement reached approximately 99%. For all test cases, it is concluded that the proposed core enhanced both the clock cycles needed to iterate one generation and the convergence speed of the existing GA implementations.

1. Introduction

Genetic algorithm (GA), is a heuristic search technique based on the natural selection process [1]. GA does not require any mathematical (analytical) solution, therefore it can successfully be applied to complex problems with wide solution space [2,3]. The main purpose of GA is to find the optimal solution in a search space by utilizing the iterative genetic evolution steps on individuals of a population. At each step, GA updates its individuals by using the genetic operations like crossover, mutation, elitism, etc. The fitness value of each individual is calculated by using an objective function of the problem. Although GAs are not evaluating all the possible solutions in the search space, time consumption of GAs increases dramatically when the problem gets more complicated. Consequently, the software based implementations of GAs are not effective for real-time applications [4]. GAs have been widely accepted as an optimization technique and used in the variety of problems including the NP-complete [5,6]. GAs have also been used in various fields like electromagnetics [7], food chemistry [8], biology, and bioinformatics [9], etc.

In recent works, hardware based solutions are used to speed up GAs [10–14]. By using a convenient design, hardware based implementations could reduce the execution time of the algorithms. Therefore the parallel design capabilities of hardware make this platform preferable for GA implementation. With the continual improvements on logic resources of field programmable gate arrays (FPGAs), it has been possible to implement efficient, customizable and fast GA architectures. Also, FPGAs are cheaper and consume low power compared to CPU and GPU based rivals. Besides, implementing evolutionary algorithms on FPGA like GAs has the advantage of producing specific hardware for special tasks [13] in the final product stage. The GAs are suitable for hardware implementation as the structure of chromosomes and genetic operators are very hardware friendly.

This paper presents the details of the proposed fast, highly customizable and extensible FPGA IP Core for general purpose genetic algorithms. The proposed architecture synthesized and tested on Altera's Cyclone EP4CE115 device. The structure of the proposed architecture is easy to integrate into different applications that need a global search approach. Also, the structure is built to enable

E-mail address: mpeker@ohu.edu.tr

https://doi.org/10.1016/j.asoc.2017.09.044
1568-4946/© 2017 Elsevier B.V. All rights reserved.

the adaptation of user-defined genetic operators and fitness function blocks. All modules of the proposed FPGA IP Core could be changed with external module blocks except the GA controller module. Therefore the proposed FPGA IP Core has the most customizable architecture in literature.

One of the best features of the proposed design is parallelization level can be selected by taking into consideration of the system resources and complexity of the fitness function. If the parallelization level increases, the system resource utilization and the speed of the system increases. Contrarily, if the parallelization level is reduced, resource utilization and the speed of the system decreases. The balance of the speed and resource usage can be adjusted with the system parameters by considering the complexity of the problem. However, the speed of the system is limited with the fitness function's computation time. Thanks to the flexible architecture of the FPGA IP Core, chromosomes of the individuals can be designed based on the problem. The proposed architecture makes it feasible to handle a wide range of problems. In this paper, the proposed FPGA IP Core is compared with recent works in literature in terms of speed, efficiency, and resource usage.

The rest of the paper is organized as follows. In Section 2, brief literature reviews for hardware based parallel genetic algorithms are given. Section 3 describes the design details of the proposed FPGA IP core structure for general purpose genetic algorithms. Section 4 gives information for the implementation platform and experimental results. Section 5 concludes the advantages and the results with a brief summary of the proposed architecture.

2. Related work

GA begins with the creation of a random population of individuals, and each individual specifies a solution in the search space. This first generation, called initial population, is developed to produce better solutions using genetic operators such as selection, crossover, mutation, etc. throughout the generations. In each generation, parents of the new offspring are selected to generate better individuals by preserving the useful knowledge of the past generations. Genetic information which is carried by the selected parents is mixed with the crossover operator to form new offspring. Then the individuals that have the worst fitness values are replaced with the new offspring to ensure the evolution to better generations. Mutation operator which randomly changes the chromosomes of individuals in the population is used to prevent the premature convergence to the best solution in the generation. Evolution process continues until the termination criteria such as the number of iterations and the fitness value of the best solution are met.

Although the GA tries to find the optimal solution without the need to compute all possible solutions, the complexity of the problem and the computational load of the genetic operators increase the run-time of the software-based implementations [4]. Therefore, in order to speed up GA-based applications, GA structures that allow parallel processing are proposed in the literature. In comparison with software-based parallel GAs, hardware-based approaches are more suitable because of the parallelization capabilities.

In the literature, many hardware based GA structures have been proposed for general [11,15] and application-specific [16–19] purposes. In this work, general purpose hardware implementations are discussed since the application-specific approaches are not applicable for other purposes.

Scott et al. [20] proposed a hardware-based genetic algorithm (HGA) which was designed with parametrized modules by using Very High Speed Integrated Circuits Hardware Description Language (VHDL) to allow scalability and provide easy reimplementation. The overall HGA design was implemented on multiple Xilinx FPGAs on a BORG board which is connected to the bus of a PC. The HGA used roulette wheel selection and one-point crossover with a fixed population size of 16. Each individual in the population had 3 bits length chromosomes. The main purpose of this work is to reveal the issues that can be experienced in hardware-based GA development.

Tommiska and Vuori [21] developed a general-purpose GA system which utilized round robin parent selection, one-point crossover and linear shift-register (LSHR) based random number generator. The population size in this work was 32 and used a PC card which is also connected to the bus of a PC to run the GA system. The GA system was used for the optimization of routing in telecommunications networks to show this system could be used in real-time applications to gain speed-up when compared to software-based approaches.

Yoshida et al. [22] proposed a VLSI hardware design for genetic algorithm processor (GAP) that uses simplified tournament selection scheme. The design is used for data partitioning problem.

Aporntewan et al. [23] implemented a compact GA using Verilog HDL. A hardware linkage learning is also proposed in [23] in order to enhance the capability to solve highly deceptive problems.

Shackleford et al. [24] presented a general-purpose pipeline GA implementation that used a survival based selection. This implementation is used for the set covering problem and the protein folding problem.

Tang and Yip et al. [25] proposed a hardware implementation of GA using FPGAs which are connected to the bus of a PC. The implementation shows that the utilization of parallel GA architectures is possible by using different communication ports such as PCI bus or on board communication ports on a PC.

Fernando et al. [15] presented a GA IP core which is customizable in terms of the population size, number of generations, crossover and mutation rates, random number generation seed and the fitness function. The design was implemented and tested on Xilinx Virtex II Pro FPGA device (xc2vp30-7ff896). The performance of this GA core was within 3.7% of the global optimal solution.

Kamimura et al. [26] demonstrated a parallel processor for distributed genetic algorithm (dGA) with redundant binary number. Their goal was to reach the optimal solution faster with redundant binary number which gives diversity. In this work, Virtex4 (xc4vlx25) FPGA board was used for the implementation of dGA.

Dos Santos et al. [17] described a cellular GA (cGa) on Virtex 6 FPGA board. In this work, GA has been built up from 5×5 array of problem-specific processing elements (PEs). They applied the cGA to a spectrum allocation problem in cognitive radio networks. The cGA was compared to a software version running on a PC and a MicroBlaze soft processor. The speedup was 95% of the PC version and 99% of the MicroBlaze soft processor.

Guo et al. [11] proposed a framework in which the GA is generated with a customization engine written in Python. The speed of the generated GA is compared with CPU based implementation of GA. The mean value of run-time enhancement was 96% of the CPU implementation. The design was implemented on Virtex 6 SX475T FPGA device.

Comparisons of these recently proposed general purpose GA designs are summarized by Fernando et al. [15]. The extended version of these comparisons with the updated literature and the proposed implementation is given in Table 1.

There are also ASIC implementations [27,10] which are focused on improving the speed of GA by using hardware acceleration. Chen et al. [27] developed a GA chip with 0.18μm cell library. A software is developed called Smart GA to change the GA network structure. Hoseini Alinodehi et al. [10] developed a CMOS implementation for steady-state genetic algorithm processor (GAP) implementation in 0.18 μm process. One GAP implementation could support a search space defined by 32-bit genetic population. They also developed