Available online at www.sciencedirect.com
ScienceDirect
Neurocomputing 71 (2007) 95–106
NEUROCOMPUTING
www.elsevier.com/locate/neucom

Hardware implementation of a novel genetic algorithm

Z. Zhu, D.J. Mulvaney*, V.A. Chouliaras

Department of Electronic and Electrical Engineering, Loughborough University, LE11 3TU, UK

Available online 3 August 2007

Abstract

This paper introduces a novel genetic algorithm whose features have been purposely designed to be suited to hardware implementation. This is distinct from previous hardware designs that have been realized directly from conventional genetic algorithm approaches. To be suitable for hardware implementation, we propose that a genetic algorithm should attempt to both minimize final layout dimensions and reduce execution time while remaining a valid implementation. Consequently, the new genetic algorithm specifically aims to keep the requisite silicon area to a minimum by incorporating a monogenetic strategy that retains only the optimal individual, resulting in a dramatic reduction in the memory requirement and obviating the need for crossover circuitry. The results given in this paper demonstrate that new approach improves on a number of existing hardware genetic algorithm implementations in terms of the quality of the solution produced, the calculation time and the hardware component requirements.

Keywords: Genetic algorithms; Genetic hardware; Machine learning

1. Introduction

Hardware implementations of a range of genetic algorithms (GAs) have been described by a number of authors with the principal purpose of reducing execution time [10,20]. Compared with their software counterparts, hardware solutions typically reduce calculation times by a factor of around 50 [2]. A further benefit of such solutions is the possibility of replicating the GA, thus allowing parallel exploration of the search space. Work in the literature includes investigations of both coarse-grained (distributed) hardware solutions [11,15], where the individual processing elements (PEs) are loosely interconnected and operate on distinct elements of the population, and fine-grained (cellular) implementations [9,13], where the PEs work collectively on the population. The latter is the main target for the current work and we specifically aim to apply modern electronic design automation (EDA) tools to generate hardware solutions that provide extremely fine division of the population between PEs. Although the architectural nature of GAs appears inherently parallel, for example, fitness calculations can be applied independently to the individuals in separate threads of computation, the application of the genetic operators frequently involves combinations of chromosomes. Consequently, a close mapping to fine-grained parallel solutions can be difficult to realize in practice. Through an investigation of novel GA architectures, the work presented in this paper is able to deliver an innovative approach to maximizing the independence between individuals; namely by completely removing the need to maintain a population.

Where the GA implementation includes a population, then, to achieve the most significant reduction in execution time, the population is best stored in on-chip memory. In such a case it is normally feasible to access the individual chromosomes at the full clock speed, but such data occupy significant physical area that could otherwise have been used for other GAs, processing elements, control devices or peripherals on the hardware system. With populations of over 100 individuals in realistic applications [7], storage of around 10–100 kB would be required. Modern field-programmable gate arrays or application-specific integrated circuits would have no practical difficulty in assimilating such memory requirements, but, as many modern implementations are system-on-chip (SoC) solutions that incorporate multi-functionality and many are deployed in portable embedded applications, it is

*Corresponding author. Tel.: +44 1509 227042; fax: +44 1509 227014.
E-mail address: d.j.mulvaney@lboro.ac.uk (D.J. Mulvaney).

0925-2312/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2006.11.031

important to avoid unnecessary use of silicon area that will not only consume power, but also could have been used by other functional components. Should such on-chip memory dominate the final layout, an alternative is to provide off-chip memory. In such cases, not only may cost considerations dictate the use of slower memory requiring a number of clock cycles to access, but also, if a number of GAs are combined in a single device, it is unlikely that the data bandwidth will be sufficient to allow all GAs simultaneous access to their respective populations. To address the memory usage issue, the compact GA [1] represented the population as a probability distribution over the set of solutions rather than requiring the storage of the entire population. The elements of a probability vector, equal in length to that of the individuals in the population, indicate the probability that the corresponding bit of an individual is unity. All vector elements are initially set to a value of 0.5. Each generation involves producing new pairs of individuals whose bit pattern is determined according to the vector probabilities. The probabilities in the vector are then modified in favor of the bit values stored in the individual of better fitness. The vector itself holds the final solution. Ramamurthy and Vasanth [10] implemented a conventional roulette-based GA in hardware. The roulette wheel was used to select pairs of individuals to be operated upon by single-point crossover, where the number of slots allotted to an individual depends on rank determined according to its fitness. Consequently, the implementation of the algorithm requires that the population is sorted in order of fitness before selection for crossover. Mutation is implemented by inverting randomly selected bits from an individual. In the implementation of Hereboy, Levi [12] combined features of simulated annealing (in that only one individual is required) and GAs (to mutate that individual). Combined with a novel method for the adaptation of the mutation rate, the approach was found to be particularly suitable for the solution of problems requiring representation by long chromosomes. As Levi considered the method particularly suited to serial rather than the parallel hardware implementations, we have chosen not to consider Hereboy as one of our alternative GAs in this paper; yet it is conceivable that minor modifications to the algorithm would allow it to become a suitable candidate solution for future investigation.

While addressing the performance and memory usage issues, it is important to ensure that any hardware implementation does not sacrifice the quality of the resulting solution. Indeed, the mere process of designing a hardware solution can almost inadvertently facilitate the development of new methods that either would not have come to light or would have been unrealistic in purely software approaches. As an example, Sharawi et al. [21] developed a crossover mechanism based on a 'half-siblings-and-a-clone' approach that was able to shorten significantly the GA convergence time. In the approach, chromosomes that best meet the fitness criteria are kept in a subsequent generation, while others are replaced by individuals that surpass a threshold generated following crossover. The crossover rate is lowered as more individuals satisfy the threshold, whose value is effectively the mutation rate.

Software implementations generally allow significant flexibility, not only in the modification of fitness calculations to meet the application requirement, but also in terms of the ease with which parameters, such as the population size, the lengths of individuals and the rate of application of operators, can be varied. These often need to be set once initially or varied during the search, for example, according to the perceived or estimated size of the solution space, the current progress of the evolution or the required diversity in the population. To permit their use in a wide range of applications, hardware implementations of GAs also need to be flexible in their structure to allow for such parameter variations [24]. Ideally, a facility to allow parameters to be set should be available prior to each application of the GA, but in practice, as a minimum requirement, it should be possible to specify such parameters at the design stage before it is progressed through the EDA flow. The more major the effects of these parameter changes, the longer the time it will generally take to generate a new hardware solution. As memory blocks typically occupy substantial silicon area, their reconfiguration often involves substantial redesign effort. Consequently, most hardware solutions do not tailor the memory requirement to the application, but assume a worst-case usage, thereby wasting significant silicon area and consuming additional power in many applications.

This paper introduces the optimal individual monogenetic algorithm (OIMGA). It is specifically designed to address the issues discussed above and achieves the following:

- Compared with conventional GAs, the memory requirement is substantially reduced since only two individuals need to be kept in on-chip registers.
- The memory requirement of OIMGA is largely independent of the application.
- The solution has the potential for dynamic reconfiguration according to the problem at hand.
- In comparison with a range of existing GA hardware methods, its performance on benchmark problems is shown to exhibit an improvement; in some cases a significant one.

The paper is organized as follows. The OIMGA algorithm is introduced in Section 2 and its hardware implementation is described in Section 3. Section 4 presents results that compare, for four hardware GA implementations, the qualities of the solution produced, the calculation times and the hardware component requirements. Conclusions that discuss the benefits of OIMGA and the planned future developments can be found in Section 5.