Microprocessors and Microsystems 58 (2018) 1–12

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

An FPGA array for cellular genetic algorithms: Application to the minimum energy broadcast problem

Pedro Vieira dos Santos*, José Carlos Alves, João Canas Ferreira

INESC TEC - INESC Technology and Science and FEUP - Faculty of Engineering, University of Porto, Porto, Portugal

ARTICLE INFO

Keywords:
Genetic algorithms
FPGA
Array processor
High-level synthesis

ABSTRACT

The genetic algorithm is a general purpose optimization metaheuristic for solving complex optimization problems. Because the algorithm usually requires a large number of iterations to evolve a population of solutions to good final solutions, it normally exhibits long execution times, especially if running on low-performance conventional processors. In this work, we present a scalable computing array to parallelize and accelerate the execution of cellular GAs (cGAs). This is a variant of genetic algorithms which can conveniently exploit the coarse-grain parallelism afforded by custom parallel processing. The proposed architecture targets Xilinx FPGAs and was implemented as an auxiliary processor of an embedded soft-core CPU (MicroBlaze). To facilitate the customization for different optimization problems, a high-level synthesis design flow is proposed where the problem-dependent operations are specified in C++ and synthesised to custom hardware, thus demanding of the programmer only minimal knowledge of low-level digital design for FPGAs. To demonstrate the efficiency of the array processor architecture and the effectiveness of the design methodology, the development of a hardware solver for the minimum energy broadcast problem in wireless ad hoc networks is employed as a use case. Implementation results for a Virtex-6 FPGA show significant speedups, especially when comparing to embedded processors used in current FPGA devices.

1. Introduction

Genetic algorithms (GAs) are optimization procedures inspired by the principles of evolution of living species, like genetics and natural selection. These algorithms implement heuristic search procedures in huge and complex solution spaces, mimicking the way living organisms evolve through generations by combining their genetic material, while promoting the survival of the best adapted to the environment. The organisms and their genetic information represent solutions of an optimization problem, the evolution of those solutions imitates reproduction by combining genetic information from different individuals, and the ability of the newborns to survive is dictated by an objective function specific to each problem.

The algorithm has become a widely accepted search metaheuristic as it does not depend on any particular characteristic of an optimization problem. Besides, complex NP-hard optimization problems with huge solutions space can only be tackled with metaheuristic search procedures to find good, although sub-optimal, solutions. Several practical applications have been addressed with genetic algorithms, like clustering in data mining and bio-informatics [1], path planning for autonomous navigation [2], antenna design [3], or energy minimization in wireless ad-hoc networks [4].

The genetic algorithm establishes only the genetic-like evolution process that must be followed to evolve and evaluate a population of solutions. The key to success when implementing the algorithm for a new optimization problem is determined by the way the solutions are encoded and combined to generate new solutions, and also the method to efficiently evaluate an objective function (or *fitness*) that guides the evolutionary process towards good solutions. These are thus very problem-specific operations and, in general, the encoding of solutions can be far more complex than just binary strings usually considered in basic implementations of genetic algorithms. In addition, the process of combining the data encoding of two solutions (the parents) to create a new solution (the child) may be far more complex than just joining parts of the solution data from the parents and may require problem-specific processing tasks to preserve the feasibility of solutions.

Although GAs can effectively explore huge solution spaces by only examining a small fraction of it, the ability to converge to good solutions relies on the creation of large numbers of solutions (or *generations*) through the repeated application of the simulated evolutionary process. This usually translates into long execution times, which can constrain the practical utilization of this metaheuristic, especially if used under

* Corresponding author.
E-mail addresses: pedro.vieira.santos@fe.up.pt, dee07008@fe.up.pt (P.V. dos Santos), jca@fe.up.pt (J.C. Alves), jcf@fe.up.pt (J.C. Ferreira).

https://doi.org/10.1016/j.micpro.2018.01.006
Received 30 December 2015; Received in revised form 6 September 2017; Accepted 29 January 2018
Available online 31 January 2018
0141-9331/

tight timing constraints or when running in embedded platforms with limited computing power.

In this work we have developed a scalable framework to support the implementation of custom computing machines in field-programmable gate arrays devices (FPGAs) for accelerating the execution of genetic algorithms. To exploit features found in modern FPGA devices, namely blocks of dual-port RAM, and allow the parallelization of the algorithm, we have addressed a particular category of genetic algorithm called *cellular*. This allows spreading the set of solutions under evolution (the *population*) over several independent memories, shared by an array of identical processing elements that concurrently implement the genetic evolution on partially overlapping sets of solutions. As a result, the performance of the computing array is almost directly proportional to the number of processing elements.

The developed computing framework provides an array of custom-designed problem-specific processing elements (PE) connected to local memory blocks shared among the PEs. To facilitate the implementation of the processing elements, a high-level synthesis design flow is proposed, where all the problem-specific operations are specified in synthesizable C++ with the help of a generic template and a library of common functions.

To demonstrate the effectiveness of the proposed computing array and design methodology, we present in this paper the design process and implementation results for an optimization problem in wireless radio networks: the minimum energy broadcast problem. This use case provides a realistic scenario where an embedded system might be required to run a genetic algorithm multiple times, as the wireless sensor network topology may change frequently because the nodes are mobile sensor devices. This is an example where it would be desirable to solve this optimization problem in embedded computing platforms instead of using desktop or higher performance computers. The cellular genetic array was successfully implemented in a Virtex-6 FPGA and exhibit relevant global speedups between 6 and 900 ×, when compared to software implementations running in embedded processors (ARM Cortex A9 and MicroBlaze).

The rest of the paper is organized as follows. Section 2 presents a brief introduction to genetic algorithms and reviews related works on custom hardware architectures for accelerating genetic algorithms. In Section 3 the cellular genetic algorithm processor (cGAP) is presented, together with the design flow based on high-level synthesis. The minimum energy broadcast (MEB) problem is introduced in Section 4, including the presentation of the variant of a genetic algorithm implemented and details of the mechanisms to encode and evolve the solutions. The hardware implementation and results are discussed in Section 5 and the paper concludes with Section 6, including proposals for further developments.

2. Related work

2.1. The genetic algorithm

The genetic algorithm (GA) is a population-based metaheuristic where a set of feasible solutions of the optimization problem to be solved, goes through an evolutionary process inspired in the biological evolution of living species [5]. The algorithm starts with a random initial population P and proceeds by executing an iterative process where genetic-inspired operations are repeatedly applied (Fig. 1).

First, a *selection* of solutions in P is performed to designate a set of solutions that will undergo some transformations to create new solutions. The selected solutions are normally called *parents* and are combined (usually in pairs) to generate new ones through a *crossover* operation that combines information from both parents. Then, the new solutions (*children*) may suffer a *mutation* operation that induces small changes to them. The generated solutions P' are then evaluated by a *fitness* function and the population for the next generation is elected among the solutions in P and P', according to some replacement strategy that takes into account the fitness values. This mechanism is the key of the iterative process as it promotes the evolution towards a better population. This process is repeated and the algorithm stops when a given criterion is met, for example, when a certain number of generations is generated without improvement of the fitness function.

While most implementations of genetic algorithms use a single population and a global selection procedure (called *panmictic*), the decentralized genetic algorithm splits the whole population into sets of solutions, or *sub-populations*, that evolve autonomously (Fig. 2). To promote the propagation of the genetic information throughout the whole population, the distributed GA periodically forces solutions to be moved between sub-populations. In the *cellular* GA the solutions are distributed along a regular grid and one solution can only interact with the solutions within a certain neighbourhood, thus creating partially overlapping sub-populations. The cellular genetic algorithm used in this work belongs to this category and the decentralization of the whole population into partially overlapping sub-populations in a key factor for exploiting the coarse-grain parallelism afforded by the proposed processor array architecture.

2.2. Hardware implementations of GAs

A continuous research activity has been carried out mainly over the last 20 years to implement custom machines for accelerating the execution of genetic algorithms by exploiting the potential of custom computing using FPGA devices. Several attempts have been made to build a generic accelerator engine for genetic algorithms, only accelerating the genetic operators and working on binary encodings of the solutions [6–8]. However, it is often the case that a GA requires a special and more complex representation of solutions, besides the basic binary codification, as for example to encode a valid paths in a graph. Additionally, some optimization problems may also require the application of specific procedures to avoid unfeasible solutions that may be generated by the application of the genetic operators. Also, the fitness function is always specific of each optimization problem and therefore cannot be efficiently handled by a general purpose GA accelerator.

Most of the works target the panmictic GA. Two different approaches can be used, depending on the way the population is evolved along the iterative process. The *generational GA* replaces the whole population in each iteration and can thus exploit the operation of several units working in parallel, accessing the whole population and generating new solutions [9,10]. However, memory access bandwidth may introduce an important bottleneck when various concurrent processing nodes must compete for the access to a single population memory. Alternatively to the generational GA, the *steady-state GA* only creates a new solution per iteration. Although the operations of this variant of the algorithm cannot be parallelized as in the generational GA, this approach has led to the proposal of efficient pipelined architectures [7,11,12].

Contrasting to the panmictic GA, in a decentralized GA the whole set of solutions is organized into sub-populations that can evolve concurrently, although a mechanism must exist to ensure some degree of interaction among the sub-populations. As referred before, this can be accomplished by periodically migrating solutions among the sub-populations, as in the distributed GA, or by using partially overlapping sub-populations, as in the cellular GA [13]. These variants of the genetic algorithm are thus the most promising for hardware acceleration, because the heuristic search procedure can be implemented as a set of concurrent processes, each one evolving its own local pool of solutions. This strategy has been adopted by various authors, presenting results that clearly show the potential of acceleration of FPGA-based custom implementation of decentralized genetic algorithms.

Table 1 summarizes the main characteristics of relevant works addressing the implementation of custom hardware accelerators for genetic algorithms. It is worth noting that a comparison between different works is not straightforward. Not only the optimization problems