Computers and Electrical Engineering 39 (2013) 2454–2468

Contents lists available at ScienceDirect

Computers and Electrical Engineering

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

FPGA implementation of particle swarm optimization for Bayesian network learning ☆

Matthew J. Hibbard [a], Eric R. Peskin [b,*], Ferat Sahin [c]

[a] SRC, Inc., 6225 Running Ridge Road, North Syracuse, NY 13212, USA
[b] Center for Health Informatics and Bioinformatics, New York University Langone Medical Center, 227 East 30th Street, #7-37, New York, NY 10016-8203, USA
[c] Department of Electrical and Microelectronic Engineering, Kate Gleason College of Engineering, Rochester Institute of Technology, James E. Gleason Hall, 79 Lomb Memorial Drive, Rochester, NY 14623-5603, USA

ARTICLE INFO

Article history:
Received 17 July 2013
Accepted 19 July 2013
Available online 26 September 2013

ABSTRACT

Using a *Bayesian network* (BN) learned from data can aid in diagnosing and predicting failures within a system while achieving other capabilities such as the monitoring of a system. However, learning a BN requires computationally intensive processes. This makes BN learning a candidate for acceleration using reconfigurable hardware such as *field-programmable gate arrays* (FPGAs). We present a FPGA-based implementation of BN learning using *particle-swarm optimization* (PSO). This design thus occupies the intersection of three areas: reconfigurable computing, BN learning, and PSO. There is significant prior work in each of these three areas. Indeed, there are examples of prior work in each pair among the three. However, the present work is the first to study the combination of all three. As a baseline, we use a prior software implementation of BN learning using PSO. We compare this to our FPGA-based implementation to study trade-offs in terms of performance and cost. Both designs use a master–slave topology and floating-point calculations for the fitness function. The performance of the FPGA-based version is limited not by the fitness function, but rather by the construction of *conditional probability tables* (CPTs). The CPT construction only requires integer calculations. We exploit this difference by separating these two functions into separate clock domains. The FPGA-based solution achieves about 2.6 times the number of fitness evaluations per second per slave compared to the software implementation.

1. Introduction

A *Bayesian network* (BN) [1] is a graphical model that expresses probabilistic relationships between variables. BNs can be used to model complex real-world systems. If the underlying relationship between variables in a system is not yet known, the structure of a BN can be *learned* from observed data. BN learning uses a function that assigns a score to each candidate network, a definition of the search space, and a search algorithm for optimization [2]. The approach to finding the best structure for a BN is an NP-Hard problem [3] and, therefore, is computationally expensive, because the number of possible structures is super-exponential in the number of variables. Because of this problem, a number of search strategies have been introduced including deterministic heuristics [4], *genetic algorithms* (GAs) [5], and *particle-swarm optimization* (PSO) [6]. Salman et al. [7] show that PSO outperforms GA with respect to convergence rate and score of a network. In this paper, we focus on PSO.

☆ Reviews processed and approved for publication by Editor-in-Chief Dr. Manu Malek.
* Corresponding author. Tel.: +1 212 263 2287.
E-mail addresses: mhibbard@srcinc.com (M.J. Hibbard), eric.peskin@nyumc.org (E.R. Peskin), feseee@rit.edu (F. Sahin).

0045-7906/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.compeleceng.2013.07.018

PSO is a method for the optimization of multidimensional continuous nonlinear functions. Each potential solution to the function is represented by a position in a predefined search space. PSO explores the search space using position and velocity equations based on previous knowledge and randomness. These equations give the swarm a diverse response. Sahin et al. [2] and Yavuz et al. [8] combine BN learning with PSO for fault diagnosis. Using a master–slave topology, the slave calculates how well a potential BN solution matches the data set using a fitness function, and the master then moves the position of the solution within the search space. Calculating the fitness of a potential BN solution requires a significant amount of time for processing in relation to the rest of the BN learning algorithm.

Hardware programming in *field-programmable gate arrays* (FPGAs) can dramatically improve the speed of an implementation by taking advantage of processes that can be computed in parallel. FPGAs are a type of reconfigurable hardware that can achieve better performance than software while allowing more flexibility than hardware [9]. A FPGA can be re-programmed multiple times by altering configuration bits. This allows a significant amount of flexibility to support many applications [10,11]. There has been prior work, in particular, in improving BN learning [12] and PSO [13] performance on FPGAs. Pournara et al. [12] show that the performance of BN learning can be increased significantly, but the size of the network learned is constrained by the size of the FPGA used. Asadi et al. [14,15] present a reconfigurable system for BN learning that achieves four orders of magnitude speedup over software for real-world networks. Their implementation relies on a pre-processing step that uses supervised learning to greatly reduce the search space. However, Linderman et al. [16] present an implementation using both FPGAs and *graphics-processing units* (GPUs) that achieves comparable performance to the FPGA system of [14] at a fraction of the cost.

Reynolds et al. [13] use two FPGAs to decrease the time it takes a PSO algorithm to complete a set number of iterations, where one FPGA is used for position and velocity equations to update the swarm, and the other calculates the fitness. Evaluating the update equations of the swarm takes little time compared to evaluating the fitness function. Reynolds et al. [13] state that using chips that are not identical could greatly increase the speed of execution of the PSO even further by giving the fitness function more resources and better hardware than the update equations.

In this present work, a hardware design of BN learning using PSO as the search algorithm is implemented on a FPGA. The starting point for the FPGA-based implementation is the software implementation of Sahin et al. [2,8]. Sahin et al. take advantage of the parallel nature of PSO by using a master–slave topology in which the slaves work independently of each other and perform the fitness computations necessary to determine how well a BN fits a given data set. Each slave is considered a particle in the swarm. The master makes the decision to keep or reject each returned score and stores the best score of each slave. The particle swarm update equations are evaluated, and the new positions are sent to each slave. This master–slave topology is retained in the hardware design. It takes a significant amount of time to process each network and calculate the fitness so that the master can make a decision and start the next iteration. For this reason, the slave is first placed into hardware, and for completeness, the master is also implemented.

The rest of this paper is organized as follows. Section 2 gives a brief description of BNs and BN learning. Section 3 describes PSO. The baseline software implementation is described in Section 4. Section 5 describes the hardware design and implementation on a FPGA. Section 6 presents the test case, methodology, and results. Section 7 presents our conclusions.

2. Bayesian networks

A BN is a *directed acyclic graph* (DAG), which is a set of directed edges between nodes where no path begins and ends with the same node [17]. The arcs between the nodes represent causal relationships between them. Each node in a BN represents a variable with a finite number of states, and the directed edges represent relationships between these variables. Fig. 1 shows an example of a BN structure. If there is a directed edge $A \rightarrow B$, then A is a *parent* of B, and B is a *child* of A. Each variable that has parents has a conditional probability table (CPT) associated with it. The CPT contains the probability of that particular variable taking a certain state given the state of its parents [8]. If a variable does not have any parents, then it has a *unconditional probability table* (UPT) instead of a CPT [1].

BNs can be applied to real-world applications with probabilistic relationships, including fault diagnosis in airplane engines, forecasting, and automated vision to name just a few [8,18]. In order to have practical use of the BN, its structure

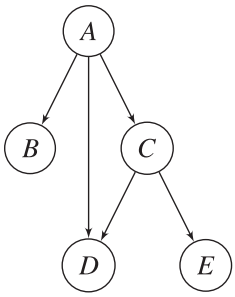

Fig. 1. Example BN.