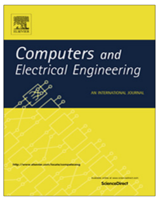
Computers and Electrical Engineering 46 (2015) 112–122

Contents lists available at ScienceDirect

Computers and Electrical Engineering

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



Use of graphics processing units for automatic synthesis of programs [☆]

Cleomar Pereira da Silva [a,b,*], Douglas Mota Dias [a], Cristiana Bentes [c], Marco Aurélio Cavalcanti Pacheco [a]

[a] Department of Electrical Engineering, Pontifical Catholic University of Rio de Janeiro, Brazil
[b] Federal Institute of Education, Science and Technology – Catarinense (IFC), Brazil
[c] Department of System Engineering, State University of Rio de Janeiro, Brazil

ARTICLE INFO

Article history:
Received 6 November 2014
Received in revised form 8 April 2015
Accepted 9 April 2015
Available online 28 April 2015

Keywords:
Genetic programming
GPU acceleration
Machine code
Quantum-inspired algorithms
Massive parallelism

ABSTRACT

Genetic programming (GP) is an evolutionary method that allows computers to solve problems automatically. However, the computational power required for the evaluation of billions of programs imposes a serious limitation on the problem size. This work focuses on accelerating GP to support the synthesis of large problems. This is done by completely exploiting the highly parallel environment of graphics processing units (GPUs). Here, we propose a new quantum-inspired linear GP approach that implements all the GP steps in the GPU and provides the following: (1) significant performance improvements in the GP steps, (2) elimination of the overhead of copying the fitness results from the GPU to the CPU, and (3) incorporation of a new selection mechanism to recognize the programs with the best evaluations. The proposed approach outperforms the previous approach for large-scale synthetic and real-world problems. Further, it provides a remarkable speedup over the CPU execution.

1. Introduction

The idea of enabling the computer to automatically create programs that solve problems establishes a new paradigm for developing reliable applications. The field of genetic programming (GP) has demonstrated that devising computer programs on the basis of a high-level description is viable.

Genetic programming extends conventional evolutionary algorithms to deal with computer programs. The essence of GP is to use the Darwinian principle of natural selection in which a population of computer programs is maintained and modified, according to genetic variation. A GP system progresses toward a solution by stochastically transforming populations of programs into better populations of programs until a stopping criterion is met.

In the past few years, GP has been successfully applied to a wide variety of problems, including automatic design, pattern recognition, financial prediction, robotic control, data mining, image processing, and synthesis of analog electrical circuits [1–7]. However, a major drawback of GP is that the search space of candidate programs can become enormous. For example, to solve the 20-bit Boolean multiplexer problem, a total of 1,310,720,000 candidate programs have to be evaluated [8]. In

[☆] Reviews processed and recommended for publication to the Editor-in-Chief by Associate Editor Dr. Jesus Carretero.
[*] Corresponding author at: Federal Institute of Education, Science and Technology Catarinense (IFC), Brazil.
E-mail addresses: cleomar.silva@ifc-videira.edu.br (C.P. da Silva), douglasm@ele.puc-rio.br (D.M. Dias), cris@eng.uerj.br (C. Bentes), marco@ele.puc-rio.br (M.A. Cavalcanti Pacheco).

http://dx.doi.org/10.1016/j.compeleceng.2015.04.006
0045-7906/© 2015 Elsevier Ltd. All rights reserved.

addition, the evaluation of the fitness of a single program in the search space may demand testing this program with numerous different combinations of input data. Consequently, the time required to evaluate the programs may be unreasonable. The computational power required by GP to evaluate billions of programs with hundreds or thousands of input data can be a huge obstacle to solving large real-world problems.

The last few years have witnessed remarkable advances in the design of parallel processors. In particular, graphics processing units (GPUs) have become increasingly popular. Further, their high computational power, low cost, and reasonable floating-point capabilities have made them attractive platforms for speeding up GP. Modern GPUs contain thousands of cores, and the massive parallelism provided is highly suitable to process GP in parallel.

The power of the GPU has been previously exploited to accelerate GP by using different methodologies. The compilation methodology [9–12] generates programs using the GPU high-level language, and each of these programs has to be compiled before its evaluation. The pseudo-assembly methodology [13,14] creates programs in the pseudo-assembly code of the GPU, and a just-in-time (JIT) compilation is performed for each individual before the evaluation. The interpretation methodology [15–17] interprets programs during its evaluation. The machine code methodology [18] generates programs in the GPU machine code and does not require any compilation step before evaluation.

The above mentioned methodologies have been used with different levels of success. However, the machine code methodology, called GPU machine code genetic programming (GMGP) [18], has exhibited significant performance gains over the others. Avoiding the compilation overhead without including the cost of parsing the evolved program is the key for severe reductions in the computational time. In addition, GMGP implements linear genetic programming by using a quantum-inspired evolutionary algorithm. The use of a quantum-inspired evolutionary algorithm provides a more efficient evolutionary algorithm that includes the past evaluation history to improve the generation of new programs. The linear genetic programming approach is more appropriate for machine code programs, as computer architectures require programs to be provided as linear sequences of instructions.

This work extends GMGP in two directions: First, we propose a new approach where the power of the GPU is fully exploited in the GP algorithm. Second, we assess the impact of selecting programs with the best evaluations on the best final solutions (i.e. programs). We have developed two approaches: GMGP-gpu and GMGP-gpu+. GMGP-gpu implements all the GP steps in the GPU and provides the following: (1) significant performance improvements in the GP steps and (2) elimination of the overhead when copying the fitness results from the GPU to the CPU through the PCIe bus. GMGP-gpu+ incorporates a new selection mechanism to recognize programs with the best evaluations. The new selection mechanism produces a more efficient comparison of the past population with the current population, bringing more diversity to the search.

The two proposed approaches were compared to GMGP and found to outperform GMGP for large-scale synthetic and real-world problems. The speedups ranged from 1.3 to 2.6. They also provided substantial speedups over CPU execution; the parallel GPGMGP-gpu+ executed up to 325.5 times faster than the CPU version.

The remainder of this paper is organized as follows: Section 2 presents previous work on parallel GP. Section 3 briefly introduces GP and quantum-inspired GP. Section 4 describes the GMGP system. Section 5 describes our approach to fully exploit the power of the GPU in GMGP. Section 6 presents the experimental results obtained for synthetic and real-world problems. Finally, Section 7 draws some conclusions and presents future research directions.

2. Related work

GP has been extensively used to solve a wide range of problems. The problem of a high computational burden associated with GP has been tackled with a variety of parallel processing methods: distributed algorithms [19], hardware implementation [20], and FPGA implementation [21].

Recently, the power of GPUs has been harnessed to accelerate GP processing in different ways. The first GPU implementations of GP were proposed by Chitty [9] and Harding and Banzhaf [10]. They used the compilation methodology and a tree representation for the programs. A single program was evaluated in parallel on the GPU with different input data. They obtained modest gains when only a little input data was used, and more prominent gains for larger datasets. Langdon and Banzhaf [15] proposed the interpretation methodology to evaluate programs and avoiding the compilation overhead of the previous approaches. The parallel algorithm evaluates the entire population of programs in the GPU by using the C++ RapidMind interpreter. The results indicated moderate speedups but demonstrated performance gains even for a very small input dataset. The interpreter proposed was further used for predicting the breast cancer survival rate in [16]. Robilliard et al. [17] studied the interpretation methodology as well. They implemented the interpreter in CUDA (Compute Unified Device Architecture) and avoided conditional branches. They obtained better performance gains than Langdon and Banzhaf [15]. Harding and Banzhaf [11] proposed a GPU implementation based on the compilation methodology, and a cluster of GPUs was used for alleviating the program compilation overhead. Speedups were obtained for very large datasets. Langdon and Harman [12] used the compilation methodology to automatically create an nVidia CUDA kernel. Numerous simplifications were employed. However, they could not automatically verify the speedup obtained as compared to the sequential CPU version. Cupertino et al. [13] proposed the use of a quantum-inspired algorithm to parallel process linear GP by using the NVidia PTX assembly language to create programs. They eliminated some compilation phases and observed gains for large datasets. However, the compilation overhead still dominated the execution time. Silva et al. [18] also used a quantum-inspired parallel linear GP but created programs in the GPU machine code. They compared their methodology with the existing compilation and interpretation methodologies and obtained significant performance gains.