Genetic Programming Approach for Classification Problem using GPU

Leo Willyanto Santoso
Informatics Department
Petra Christian University
Surabaya, Indonesia
leow@petra.ac.id

***Abstract*—Genetic programming (GP) is a machine learning technique that is based on the evolution of computer programs using a genetic algorithm. Genetic programming have proven to be a good technique for solving data set classification problems but at high computational cost. The objectives of this research is to accelerate the execution of the classification algorithms by proposing a general model of execution in GPU of the adjustment function of the individuals of the population. The computation times of each of the phases of the evolutionary process and the operation of the model of parallel programming in GPU were studied. Genetic programming is interesting to parallelize from the perspective of evolving a population of individuals in parallel.**

Keywords— classification, evolutionary algorithms, genetic programming, parallel

I. INTRODUCTION

Evolutionary computing is encompassed within a broad set of problem-solving techniques based on emulation of natural processes of evolution. The main contribution of evolutionary computation to the problem-solving methodology consists in the use of mechanisms for the selection of potential solutions and the construction of new candidates by recombination of character. Statistics of others already present, in a similar way to what happens with the evolution of organisms already present. It is not so much a question of reproducing certain phenomena that occur in nature, but rather of taking advantage of the generic ideas behind them. When there are several candidates for a solution to a problem, the need arises to establish quality and selection criteria and also the idea of combining characteristics of good solutions to obtain better ones. Given that it was in the natural world that problems of this type were first raised, it is not surprising that when applying such ideas in solving scientific and technical problems, procedures quite similar to those obtained were obtained. that are already found in nature after a long period of adaptation.

Within evolutionary computing is genetic programming, in which the individuals who evolve in the system are computer programs that represent, in whole or in part, the solution to the problem posed. It is therefore a machine learning method used to do optimization of a population based-on an adjustment function that evaluates the capacity of each part to carry out a task in question. A series of modifications can be made on each individual using the genetic operators in a similar way to what occurs in natural organisms. The best known formulation of genetic programming is due to John Koza, who represents individuals as instruction trees [10].

Genetic programming algorithms have been implemented to the resolution of numerous data mining problems. Data mining [2, 3, 4] is defined as the nontrivial extraction of implicit, formerly unidentified, and potentially valuable information from data. In today's information society, where the amount of data stored is multiplied almost exponentially every day, data mining is a fundamental tool to analyse and exploit it effectively. Data mining techniques provide insight from data relationships and provide researchers and users with rules for classification, association [5, 6, 7], and prediction. The usage of genetic programming in solving classification problems is relatively frequent [8, 9, 10, 11]. In this case, these are supervised learning algorithms where individuals in the population represent a classifier in whole or in part, and their evaluation measures the ability to correctly classify a dataset that has been externally evaluated [12]. The objective is that the generated classifier can be used successfully in classification of unknown patterns. In data mining and unsupervised machine learning, association rules [13] are used to determine events that occur in mutual within a given data set. Numerous approaches for discovering association rules have been extensively investigated and have been very interesting for determining relationships between features in large data sets.

In solving this type of problem, a series of drawbacks should be minimized, such as the high computational cost it has, the large number of data necessary to evaluate individuals, etc. It is because their convergence to the solution can be very slow in complex or large problems. To speed up its performance, its parallelization has been the object of study from multiple perspectives, taking advantage of the development of new parallel hardware and the different characteristics of the domains of the particular problem. Most of the parallel algorithms developed in the last two decades focus on implementation in clusters or massively parallel architectures. More lately, work on parallelization has focused on the use of graphics processing units (GPUs) that provide very fast and natively parallel hardware at a fraction of the cost of a traditional parallel system [14]. This paradigm is known as GPGPU (General Purpose Graphic Processing Unit) computing [15]. Specifically, the use of GPUs for solving genetic programming problems is called GPGPGPU [16, 17, 18].

The objective of this research is to analyse the operation and computational cost of genetic programming methods applied for classification problems with the purpose of accelerating their execution processes. For this, a study of the phases of the evolutionary algorithm will be carried out with the aim of understanding its high computational cost and an analysis of the GPU programming model that allows us to acquire sufficient knowledge to carry out an efficient design of the algorithm. The experimentation will look for that configuration of the algorithm parameters that maximize the

excellence of the produced solutions minimizing the execution time. The results of comparison of performing a tests with multiple data sets with the running time of the implemented algorithms with respect to our proposal in multithreading and in GPU. Finally, the conclusions of the project are drawn and the ways of future work are stated.

II. Literature Review

A. Genetic Programming

Genetic programming is a method derived from genetic algorithms [19]. Mainly, genetic programming eliminates the limitation that genetic algorithms have to represent solutions with a priori unknown size. The latter represent each solution with a fixed-length character string, which limits the information that a solution can contain. Genetic programming solves this obstacle by representing each solution as a hierarchically organized structure of nodes, commonly known as a tree. New nodes can be added to this structure in each generation, or existing nodes can be removed or replaced by others.

However, in practice, the space limitation of a solution cannot be eliminated. Any machine where the algorithm is executed has a limited amount of memory to be able to save the solutions of each generation, with which there is always a limit on the length of the solutions. In addition, due to the organization of memory in computers, the more the size of a solution grows, the longer it takes to access it to combine it with others or calculate its quality, which usually limits considerably the maximum space that can be used. occupy a solution (at most a few Kilobytes).

In genetic programming, individuals or solutions are represented as a hierarchical structure of nodes, that is, a tree [20]. Each node represents a function, which returns a result to the node from which it descends and has parameters given by its children. The terminal nodes (leaves) represent constants or specific data of the problem scenario. The structures are evaluated recursively from the root to the leaves, and their result is the one obtained from the root.

To build any solution, a predefined set of nodes is available, that is, functions, problem data and constants. These nodes are chosen according to specific criteria for each problem, as they must be relevant for a combination of them to produce a good quality solution. It should also be noted that any combination of these nodes must produce a valid solution, that is, the result of the evaluation of each type of node must serve as a parameter for any other. For example, if a node of type sum returns an integer and descends from a node of type AND, the latter must know how to interpret the integer returned by its descendant as true or false.

The operators used in genetic programming are the same as in genetic algorithms. There are two fundamental operators: reproduction and crossing. The breeding operator is responsible for passing an individual to the following generation unchanged, while the crossing operator is responsible for obtaining one or more individuals from two. The reproduction operator normally depends on how often the other operators are applied, since the number of individuals is usually kept constant throughout the generations, with which, the individuals that have not been altered by other operators pass without changes to the next generation. Another widely used operator, although not essential, is the mutation operator, which randomly changes a part of an individual. As the operators used and configurable in the program carried out for this project, the crossover and mutation operators will be detailed in the following sections. Other operators include: permutation, to exchange the position of two nodes of the same individual; editing, to simplify individuals by substituting subtrees that always give the same result by nodes with the constant result of the evaluation; encapsulation, to create a node that contains the functionality of a subtree of an individual; destruction, to eliminate, usually in the first generations of the algorithm, individuals with a very low quality.

Fitness function is the function that calculates the quality of an individual is essential for the correct functioning of the algorithm, since it guides the evolution of the population. The other components involved in the algorithm select individuals for the next generation depending on the value returned by said fitness function. This function emulates in reality the probabilities that an individual has of reproducing, or in other words, their expected survival time and number of offspring. However, this is not a faithful simulation of reality, since in genetic algorithms and genetic programming, a high quality individual can survive an indefinite number of generations because it could not be replaced by another. This can be avoided by adding to the fitness function a penalty according to the age or number of generations since the individual exists.

B. Related Research

Numerous works and publications have been found about the use of GPUs and massively parallel architectures in the framework of genetic programming.

A general-purpose computing technique using graphic cards and how to implement this approach to genetic programming were proposed [21]. It demonstrates the improvement in runtime performance in solving genetic programming problems on single-processor architectures. Moreover, how to specifically accelerate the evaluation of individuals in genetic programming was implemented [22].

Previous studies of GPU work demonstrate the feasibility of accelerating evaluation for large data sets. Furthermore, a parallelization scheme to take advantage of the computational potential of the GPU on smaller data sets was proposed [23]. To achieve optimization with smaller databases, instead of sequentially evaluating the individuals, parallelizing the evaluation of the training patterns, the parallelization of the GPU is also shared among the individuals. Therefore, the different programs are assessed in parallel and each is assigned to an execution unit in which to execute the training patterns in parallel.

A comparable practice but using an application based on a Single Program Multiple Data (SPMD) model was proposed [24, 25]. The practice of an SPMD model instead of a Single Instruction Multiple Data (SIMD) offers the chance to reach higher performances, for example, an execution unit can perform the if division of a conditional while another unit can execute independently the else branch. Carrying out the execution of both branches within the same implementation unit is also thinkable, but the two branches would be treated consecutively according to the SIMD model, this is named divergence and therefore it is less effective.

III. Methodology

In this research, several software resources were used for the development of the application: Windows 10 64bit, C/C ++ GNU Compiler 4.5, JAVA SE 15, NVIDIA CUDA SDK 3.1 and Eclipse 4.15. The research was performed in an AMD Ryzen 7 with 8 cores and 16 processing threads.

The process of evolution of proposed genetic programming algorithms is involves of the following phases:

1. An original population of individuals is created according to an initialization procedure.
2. It is assessed to obtain the quality of its individuals, that is, the quality of the rules is checked in terms of their ability to classify a set of data correctly.
3. In each group, the procedure selects a part of the population as parents to procreate. The selection procedure usually takes the best individuals as parents to confirm the survival of the finest genetics.
4. This subdivision of individuals is traversed applying the different genetic crossing operators, obtaining an offspring.
5. These offspring can be mutated using the dissimilar genetic mutation operators.
6. These new individuals should be evaluated using the adjustment function to get their quality index.
7. Different approaches of replacement of population individuals and parents by offspring can be applied to confirm that the population size in the next generation remains constant and maintains the best individuals.
8. The procedure performs a controller phase in which it determines whether it should end its execution because it has found solutions of acceptable quality or because it has touched a limit quantity of generations, if it does not go back to step 3 and performs a new iteration. The chained processes of parental selection, crossing, mutation, evaluation, replacement and control constitute one generation of the algorithm.

IV. Discussion and Analysis

Experiments have been carried out on 5 public domain data sets from the UCI repository for machine learning [26]. These data sets show a good diversity with respect to different characteristics, such as quantity of examples, classes, attributes, and data types. The objective is to analyse the performance of the algorithm's execution. The features of the data sets used are detailed below and summarized in Table 1.

TABLE I. Summary of the Datasets used to evaluate the GP algorithm

Datasets Name	#Instances	#Attributes
Abalone	4177	8
Chess	28056	6
Credit Approval	690	15
Iris	150	4
Mushroom	8124	22

- Abalone: Forecasting the oldness of abalone from physical measurements.
- Chess: it is a created database in a single iterative process with 6 attributes.
- Credit approval: This file contains credit card applications. All feature: names and values have been changed to worthless symbols to keep privacy of the data.
- Iris: It is the most well-known database for pattern recognition. It consists of 3 classes (Iris Setosa, Iris Versicolour and Iris Virginica) of 50 instances each representing a type of iris plant. The classifier must predict the type of the plant based on four attributes: length and width of the sepal and petal.
- Mushroom: This data set contains descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms.

The size of the population is a key factor in generating quality solutions. A greater number of individuals will allow the search space to be expanded, favouring diversity and obtaining a greater number of good solutions.

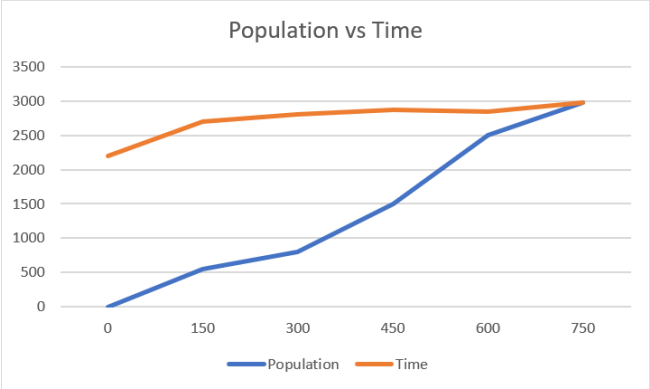

Fig. 1. Function of population size

As can be seen in Figure 1, the execution time is a linear function with respect to the population size and to obtain quality solutions a large population size is required. Typically, a population size of 400 may be sufficient, but for classifying large data sets we can scale up to 750 individuals to explore larger areas of the search space if necessary. An acceptable compromise value is around 500 individuals, this is the size of the population for the execution of the tests.

Evolutionary algorithms improve their individuals in each generation by applying crossover, mutation, and selection operators so that the best individuals move on to the next generation. A greater number of generations should offer individuals of higher quality.

Figure 2 represents that the execution time is a linear function with respect to the number of generations. However, an excessively high number of generations does not guarantee convergence to an excellent solution. Usually, with a number of generations greater than 300, good solutions are reached, so this will be the value of the parameter in the execution of the tests.

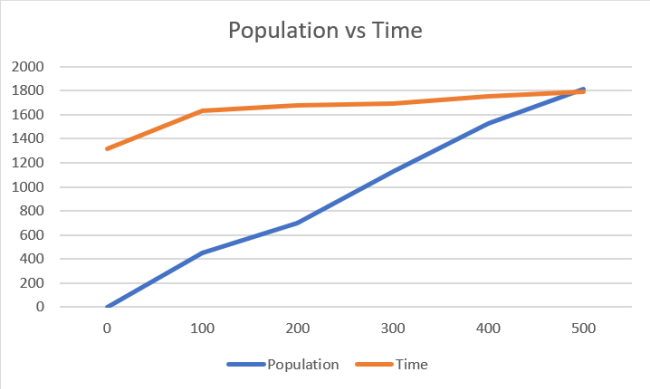

Fig. 2. Hit rate based on the number of generations

For practical criteria, the definition of convergence is very useful. If the genetic algorithm has been appropriately implemented, the population will grow over succeeding generations in such a way that the mean adaptation extended to all individuals in the population, as well as the adaptation of the best individual will increase towards the global optimum. The idea of convergence is related to the evolution towards uniformity: a gene has converged when at least 90% of the individuals in the population share the same value for that gene. The population is said to converge when all genes have converged. This definition can be generalized to the case in which at least a few of the individuals in the population have converged. One of the greatest enemies of genetic algorithms is premature convergence, since if the solution is not converged, the algorithm closes itself to exploring other areas of the search space where it can find better solutions.

It can be concluded with the experimentation of the last two parameters that determine the quality of the solutions, that to obtain a high success rate in large data sets, a large population will be required. It evolves in a high number of generations and therefore the execution time will be decisive. This is the opportunity for GPUs to show their potential and speed up the process of complex dimensional problems.

GP's algorithm has been verified with all the data sets and the following times of the evaluation phase have been obtained, expressed in milliseconds per columns in Table II and in Figure 3, for an execution with a population size equal to 500 that guarantees good solutions. Each database is tested with different configurations: the first column represents the original iterative version in Java, the second externalizes the evaluation phase in an iterative but native method written in C, in the third column the evaluation It is divided into two native threads in which each one evaluates half of the population, the fourth corresponds to the native evaluation using 4 threads for its execution in the 4 microprocessor cores, the next two reflect the execution times by externalizing the evaluation on one and two graphic cards respectively.

TABLE II. SUMMARY OF THE EXECUTION TIME TO EVALUATE THE GP ALGORITHM

Datasets	Java	C	2T	4T	1GPU	2GPUs
Abalone	8793	10452	7691	4205	2057	1032
Chess	31246	34129	27419	19250	10035	6459
Credit Approval	957	1140	751	579	273	163
Iris	415	740	340	290	127	71
Mushroom	14753	17529	12155	8305	4571	2132

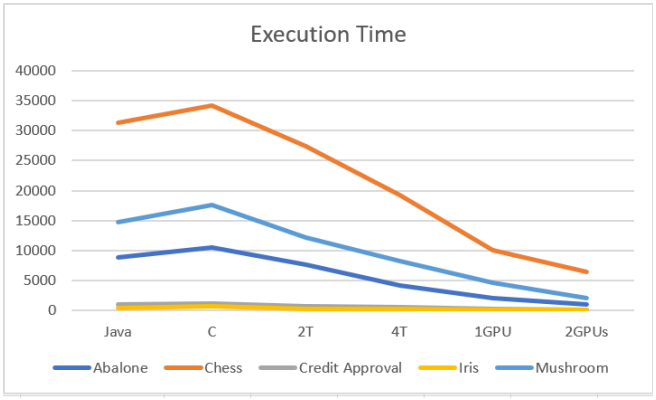

Fig. 3. Execution time of several datasets

V. CONCLUSIONS

The classification of large data sets is an operation that demands a lot of time and computational resources as the size of the problem increases.

Genetic programmings are exciting to parallelize from the view of developing a population of individuals in parallel. However, the classic view of the genetic algorithm is not fully parallelizable due to the need to have global control in certain stages such as selection and crossing that require serialization of the execution.

The parallelization of the population evaluation phase accelerates the operation of the algorithm. The use of threads allows dividing the individuals to be evaluated into subsets that can be executed in parallel in each of the microprocessor's cores. Current CPUs have up to 4 cores, so it cannot be accelerated beyond that. However, GPUs have evolved to an architecture with a massive number of cores that under a SIMD model, can execute millions of threads concurrently.

REFERENCES

[1] J. Koza, Genetic programming: On the programming of computers by means of natural evolution. Cambridge: MIT Press, 1992.

[2] L. W. Santoso, Early Warning System for Academic using Data Mining, Proceedings of the 2018 Fourth International Conference on Advances in Computing, Communication & Automation (ICACCA), 2019. https://doi.org/10.1109/ICACCAF.2018.8776788

[3] G. Ramu, M Soumya, A. Jayanthi, J. Somasekar, and K. K. Baseer, "Protecting big data mining association rules using fuzzy system," TELKOMNIKA vol 17, no. 6, December 2019. http://dx.doi.org/10.12928/telkomnika.v17i6.10064

[4] M. S. Das, A. Govardhan, and D. V. Lakshmi, "Classification of web services using data mining algorithms and improved learning model," TELKOMNIKA vol 17, no. 6, December 2019. http://dx.doi.org/10.12928/telkomnika.v17i6.11510

[5] L. W. Santoso and Yulia, "The Analysis of Student Performance Using Data Mining," Advances in Computer Communication and Computational Sciences. Advances in Intelligent Systems and Computing, vol 924. Springer, Singapore. https://doi.org/10.1007/978-981-13-6861-5_48.

[6] E. Alothali, H. Alashwal, and S. Harous, "Protecting big data mining association rules using fuzzy system," TELKOMNIKA vol 17, no. 2, April 2019. http://dx.doi.org/10.12928/telkomnika.v17i6.10064

[7] L. Zhang, L. Xu, and L. Rai, "High-precision Ultrasonic Flowmeter for Mining Applications based on Velocity-area," TELKOMNIKA vol 16, no. 1, February 2018. http://dx.doi.org/10.12928/telkomnika.v16i1.5185

[8] S. Sendari, A. N. Afandi, I. A. E. Zaeni, Y. D. Mahandi, K. Hirasawa, and H.-I. Lin, "Exploration of genetic network programming with two-stage reinforcement learning for mobile robot," TELKOMNIKA vol 17, no. 3, June 2019. http://dx.doi.org/10.12928/telkomnika.v17i3.12232

[9] M. M. Bhaskar and S. Maheswarapu, "A Hybrid Genetic Algorithm Approach for Optimal Power Flow," TELKOMNIKA vol 9, no. 2, August 2011. http://dx.doi.org/10.12928/telkomnika.v9i2.689

[10] A. Tsakonas, "A comparison of classification accuracy of four Genetic Programming-evolved intelligent structures," Information Sciences, 176 (6), 2006, pp. 691-72.

[11] C. C. Bojarczuk, H. S. Lopes, A. A. Freitas, and E. L. Michalkiewicz, "A constrained-syntax Genetic Programming system for discovering classification rules: application to medical data sets," Artificial Intelligence in Medicine, 30 (1), 2004, pp. 27-48.

[12] I. De Falco, A. Della Cioppa, and E. Tarantino, "Discovering interesting classification rules with Genetic Programming," Applied Soft Computing Journal, 1 (4), 2002, pp. 257-269

[13] C. Romero, J. R. Romero, J. M. Luna, and S. Ventura, "Mining Rare Association Rules from e-Learning Data," Educational Data Mining 2010, 2010.

[14] S. Ryoo, C. Rodrigues, S. Baghsorkhi, S. Stone, D. Kirk and W. Hwu, Optimization principles and application performance evaluation of a multithreaded GPU using CUDA, PPoPP '08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming, 2008, pp. 73-82.

[15] B. Oancea, T. Andrei, and R. M. Dragoescu. GPGPU Computing. https://arxiv.org/ftp/arxiv/papers/1408/1408.6923.pdf

[16] J. Kim, J. Kim, and S. Yoo, "GPGPGPU: Evaluation of Parallelisation of Genetic Programming Using GPGPU," In: Menzies T., Petke J. (eds) Search Based Software Engineering. SSBSE 2017. Lecture Notes in Computer Science, vol 10452. Springer, Cham. https://doi.org/10.1007/978-3-319-66299-2_11

[17] S. Yoo, M. Harman, and S. Ur, "GPGPU test suite minimisation: search based software engineering performance improvement using graphics cards," Empirical Softw. Eng. 18(3), 550–593 (2013)

[18] G. Wilson and W. Banzhaf, Deployment of CPU and GPU-based genetic programming on heterogeneous devices. In: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference (GECCO 2009), pp. 2531–2538. ACM Press, New York, July 2009

[19] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and its Applications, Morgan Kaufmann, San Francisco, 1998

[20] W. Langdon and B. Buxton, Genetic programming for combining classifiers, Proceedings of the Genetic and Evolutionary Computation Conference, 2001, pp. 6673.

[21] D. Chitty, A data parallel approach to Genetic Programming using programmable graphics hardware, GECCO'07: Proceedings of the Conference on Genetic and Evolutionary Computing, 2007, pp. 1566-1573.

[22] S. Harding and W. Banzhaf, Fast Genetic Programming and artificial developmental systems on GPUs, HPCS'07: Proceedings of the Conference on High Performance Computing and Simulation, 2007.

[23] D. Robilliard, V. Marion-Poty, and C. Fonlupt, "Genetic programming on graphics processing units," Genetic Programming and Evolvable Machines, 10 (4), 2009, pp. 447-471.

[24] W. Langdon and A. Harrison, "GP on SPMD parallel graphics hardware for mega bioinformatics data mining," Soft Computing. A Fusion of Foundations, Methodologies and Applications, 12 (12), 2008, pp. 1169-1183.

[25] W. Langdon, W. Banzhaf, A SIMD interpreter for genetic programming on GPU graphics cards. In: O'Neill, M., Vanneschi, L., Gustafson, S., Esparcia Alcázar, A.I., Falco, I., Cioppa, A., Tarantino, E. (eds.) EuroGP 2008. LNCS, vol. 4971, pp. 73–85. Springer, Heidelberg (2008). doi: 10.1007/978-3-540-78671-9_7

[26] D. Dua and E. Taniskidou, "UCI Machine Learning Repository." Internet: http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science, 2017.