Parallel Algorithm for Reduction of Data Processing Time in Big Data

Parallel Algorithm for Reduction of Data Processing Time in Big Data

Jesús Silva[1], Hugo Hernández Palma[2], William Niebles Núñez[3], David Ovallos-Gazabon[4] and Noel Varela[5]

[1]Universidad Peruana de Ciencias Aplicadas, Lima, Perú.
[2] Universidad del Atlántico, Puerto Colombia, Atlántico, Colombia.
[3]Universidad de Sucre, Sincelejo, Sucre, Colombia.
[4]Universidad Simón Bolívar, Barranquilla, Atlántico, Colombia
[5]Universidad de la Costa, Barranquilla, Atlántico, Colombia.

[1]Email: jesussilvaUPC@gmail.com

Abstract. Technological advances have allowed to collect and store large volumes of data over the years. Besides, it is significant that today's applications have high performance and can analyze these large datasets effectively. Today, it remains a challenge for data mining to make its algorithms and applications equally efficient in the need of increasing data size and dimensionality [1]. To achieve this goal, many applications rely on parallelism, because it is an area that allows the reduction of cost depending on the execution time of the algorithms because it takes advantage of the characteristics of current computer architectures to run several processes concurrently [2]. This paper proposes a parallel version of the FuzzyPred algorithm based on the amount of data that can be processed within each of the processing threads, synchronously and independently.

1. Introduction

FuzzyPred is a data mining method that allows the extraction of fuzzy predicates in normal conjunctive and disjunctive form [3] [4]. This method is modeled as a problem of combinatorial optimization because the space of solutions to travel can become very large. The algorithm in charge of evaluating the quality of each predicate has a polynomial temporal complexity of O (t*k*v), where *t* is number of records, *k* is number of clauses, and *v* is number of variables) in the worst case. Each generated solution (or predicate) is sequentially evaluated in each of the database records. Considering the above, and due to the fact that the dimensions and the number of variables of the current databases increase in size every day, it is possible to obtain high response times in this process by using FuzzyPred [5].

Because parallel computing must be exploited to solve data mining problems, this paper presents a parallel version of FuzzyPred with the purpose of reducing runtime. The fundamental objective of the applied design is to perform a parallel processing focused on the database size, where the hardware potentials that exist today can be used in a flexible way. In the studies, experiments are performed to compare the sequential version with the parallel version of FuzzyPred, in different performance metrics (particularly acceleration and efficiency).

2. Materials and Methods

2.1 Parallel Algorithm Design

Today, many problems require companies to process large amounts of data and make the response time efficient in their applications. In this sense, the computer efficiency depends directly on the time required to execute a basic instruction and the number of instructions that can be executed at the same period of time [6]. Thus, parallel programming is an area of computing that takes advantage of hardware resources to improve algorithm execution times.

In parallel programming, there are two types of parallelism [7]: control parallelism (functional decomposition) or data parallelism (domain decomposition). The domain decomposition or data parallelism, as it is also known, consists of a sequence of instructions applied to different data. The data is divided into parts and the parts are assigned to different processors. Each processor works only with the part of the data that is assigned and the processors may need to communicate to exchange the data. Data parallelism allows maintaining a single control flow and following the Single Multiple Data Program (SMTP) model [8].

In functional decomposition or task parallelism (also dynamic task distribution), the problem is divided into a large number of smaller parts (many more parts than available processors) and the sub-tasks are assigned to available processors. As soon as a processor completes a sub-task, it performs another sub-task until all of them are finished. The task parallelism is applied on a master and slave paradigm. The master process assigns the tasks to the slave processes, collecting the results produced and assigning remaining sub-tasks [9]. In recent years, due to the increase in the scale of parallelism that is necessary in some situations, the term Big Data has come to be coined, with its consequent conceptual and technological framework [10].

A sequential algorithm essentially follows a sequence of steps to solve a problem using a single processor. Similarly, a parallel algorithm follows and solves the sequence of steps using multiple processors. Parallel algorithms are designed in such a way that several of these steps can be solved concurrently. It is essential, in order to obtain any benefit from the use of parallel computers, to have a good algorithm design [11].

In practice, it is not trivial to perform this design, so a set of steps (some or all of which may be included) are followed for the design of a parallel algorithm, which are discussed below [12]:

1. Identify the parts of the algorithm that are most costly and can be executed concurrently.
2. Map the parts that can be executed concurrently within multiple parallel processes.
3. Distribute input, output and intermediate data in the program.
4. Allow data access to multiple processors.
5. Synchronize multi-level processes in the execution of the parallel program.

2.2 FuzzyPred

FuzzyPred is a data mining method that proposes fuzzy predicates in a normal conjunctive and disjunctive way as a way of representing knowledge. This method solves a descriptive task where the types of relationships are unknown, and looks for patterns that describe the data and their relationships. This method is modeled as a combinatorial optimization problem because the space of solutions through which it can transit can become immense [13] [14].

2.3 Analysis of the main FuzzyPred processes

In order to optimize the execution time of FuzzyPred, a study was carried out about the main processes which need, from a computational point of view, more features as they have more workload and take longer to execute. Due to their characteristics, the identified processes were: the evaluation of each predicate and the post-processing stage of the results.

The predicates evaluation process interacts with the entire data system. In this process, the key is to consider the size of these systems. It is important for this process to emphasize that, as a consequence of all the accumulation of information that is currently available, databases can become immense, so parallelizing this process is a fundamental issue for the proper functioning of FuzzyPred, as long as it runs on a computer that has several threads [15].

The post-processing stage, on the other hand, has as its main objective to offer a more readable set of predicates for the user's comprehension and is formed by four main methods: eliminating repeated predicates, eliminating equal clauses, decreasing variables and eliminating obvious predicates. These functions interact with all the results obtained by FuzzyPred, and need to process the structure of each predicate, and compare, in most of the cases, with the remaining predicates in the set. One of the challenges of data mining today is the large number of solutions that can be provided by each of the algorithms [16]. In the specific case of FuzzyPred, the number of predicates obtained can be very large, even when the databases are not so large, since the space of solutions that can be covered is enormous because the number of variables of the problem increases, the latter being a significant element since the study deals with linguistic labels and not with the real attributes of the databases.

In the case of the post-processing stage, a functional parallelization model is not applied since these functions are dependent on each other. They were created with a definitive and inviolable order since the output values of one function represent the input values of the next one [17].

To analyze each of these processes, their algorithmic complexity was taken into account. Algorithmic complexity [18] represents the amount of time resources needed by an algorithm to solve a problem and therefore allows the efficiency of that algorithm to be determined. The criteria to be used to assess algorithmic complexity do not provide absolute measures but measures of the problem size.

In the evaluation process there are nested cycles, the analysis of each one will be carried out from inside out. The first step of the algorithm is to evaluate each variable in each of the clauses, this process has a complexity of O (1) for each variable, so for the whole set it would be a complexity O (v), where v represents the number of variables of a clause. The second process is to evaluate each clause of a predicate. For this process, it considers whether the predicate is in FNC or FND and the complexity is O (k) * (O (v) + O (v)) = O (k) * max (O (v), O (v)) = O (k * v) where k represents the number of clauses. The third process is to evaluate the predicate for each record of the database. For this cycle, it also considers the structure of the predicate and the order is O (t) * (O (k *v) + O (k)) = O (t) * max (O (k *v), O (k)) = O (t * k * v) where t represents the number of records in the database (Taymi, 2010). The execution time of a sequence of instructions is equal to the sum of their individual execution times, which is equivalent to the maximum order. In FuzzyPred, it is: max (O (t *k *v), O (t)) = O (t *k *v). This execution time can be considerable since the factors that influence it can also take great values. This is why the following section is based on the proposed parallelization of this algorithm specifically in the process of evaluating fuzzy predicates [19] [20].

3. Parallel design proposal in FuzzyPred as a solution to high data dimensionality

For the parallelization design of FuzzyPred specifically in the evaluation process, the of data parallelism paradigm is applied, basically to the database to be used in the mining process.

In this design, the evaluation of the predicate was carried out in each part of the data system independently and simultaneously. To do this, a set of steps is followed, which are discussed below:

1. At the beginning, the number of threads of the computer's processor is known. The Java Parallel library performs this process in a scalable way.
2. Subsequently, groups are created depending on the number of threads contained in the architecture where the algorithm is executed.
3. Consequently, the created groups, the threads of execution are assigned in a dynamic way looking for that all the threads have the same amount of work.
4. Finally, a barrier was used to carry out the evaluation applying the universal quantifier, since it needs to know all the truth values of the predicate in each of the records of the database.

4. Results

This section presents the validation of the proposed solution. For this purpose, performance metrics are applied to parallel algorithms and a series of comparative tests are developed. The objective of this section is to verify that the execution time of the parallel algorithm decreases with respect to its sequential version.

For the experiments, several databases were considered, with different characteristics (shown in Table 1). These databases are from the real environment and were taken from the UC Irvine Machine

Learning Repository, which offers the researchers a wide range of data collected from different areas. The chosen databases have different sizes with the purpose of valuing this characteristic.

Table 1. Description of the databases used in the experimentation

Names	# Records	# Attributes (R/N/E/)	# Linguistic Labels
Quacke	2965	3 (3/1/0)	8 (8/0/0)
Stulong	2014	7 (7/0/0)	16 (16/0/0)
Bolts	60	10 (2/5/3)	18 (18/0/0)

The algorithm was tested in Java under the Eclipse development environment, compiled with JDK 1.7. To analyze the behavior of the parallel algorithm, three scenarios were designed with different objectives. The aim of the first scenario is to compare the sequential version with the parallel version of FuzzyPred. Thus, both versions were run on the same computer with the same hardware performance and under the same input configuration of FuzzyPred (same algorithm input parameters and in the same database). These parameters are presented in Table 2.

Table 2. Scenario 1 Configuration Parameters

FuzzyPred Parameter	fuzzy logic operator=zadeh, connective=random, value scale=0-14, runs=50, iterations per run=10000, target=truth value, metaheuristic algorithm=random search, database=Quacke.
PC Characteristics	Intel Core i3 -2100 CPU 4 Gb de RAM

As shown in Table 4, the runtime (in minutes) of FuzzyPred was taken in both versions (sequential and parallel). It is important to consider that although databases do not have a large number of records (which represents a negative factor since the parallel proposal may not show improvement), the parallel execution time improves the sequential execution time by 10%, demonstrating an improvement for this last version.

The results achieved in this experiment are as follows:

Table 3. Sequential vs Parallel Version Run Times

Quality Metrics		Value
Execution time	**Sequential time**	84 min
	Parallel time	77 min
Speed-up		1.20
Efficiency		0.14

The aim of the second scenario is to compare the parallel version of FuzzyPred against several computers with different characteristics and to know its behavior in different hardware environments. Table 4 shows the configuration parameters of FuzzyPred and Table 5 shows the characteristics of each of the computers on which the experiments are run.

Table 4. Scenario 2 Configuration Parameters

FuzzyPred Parameters	fuzzy logic operator=zadeh, connective=random, value scale=0- 14, runs=50, iterations per run=10000, objective=truth value, metaheuristic algorithm=random search, database=Quacke.

Table 5. Characteristics of the hardware applied in scenario 2

Hardware	Memory		
	Number of processors	**Type**	**Quantity (Mb)**
Intel Core 2 Duo E7300	2 nuclei	DDR2	2369
Intel (R) Core TM 2 Quad Q9300	4 nuclei	DDR2	5874
Intel Core i7 920	8 nuclei	DDR 3	2036

Subsequently, Table 6 shows the results achieved in each of the hardware architectures with respect to parallel runtime, sequential runtime, and values for acceleration and efficiency metrics.

Table 6. Results obtained from the parallel version in different hardware.

Hardware	Sequential T.	Parallel T.	Acceleration	Efficiency
Intel Core 2 Duo	130 min	85 min	1.21	0.77
Intel (R) Core TM 2 Quad Q 9300	961min	58 min	1.47	0.52
Intel Core i7 920	69 min	39 min	1.75	0.35

The results in scenario 2, reflected in Figure 1 regarding the execution time show that the best values are found in Intel Core i7 architecture, due to the fact that it is the one with the best computing performance. However, Figure 2 shows the results in each of the measures considered in this study, where it can be observed that acceleration increases as it improves the characteristics of the hardware, contrary to efficiency, which decreases because the design is capable of exploiting the characteristics of the hardware.

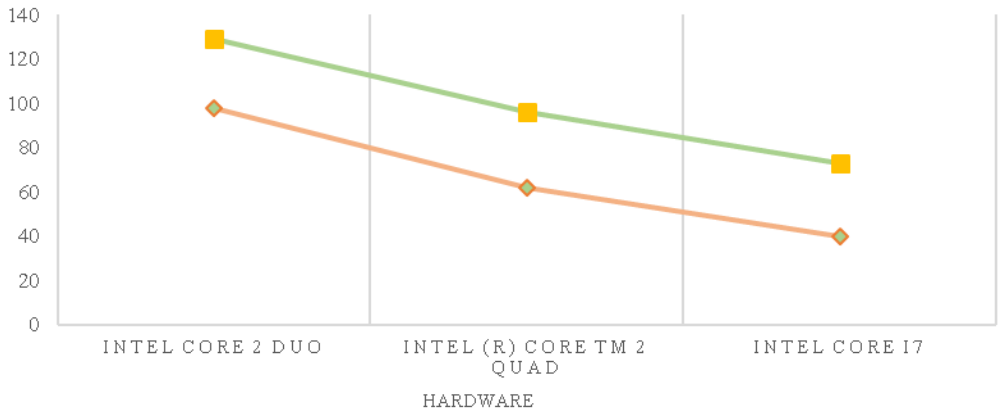

Figure 1. Sequential and parallel runtime behavior for various hardware architectures. The parallel time is in red, and the sequential time is in green.

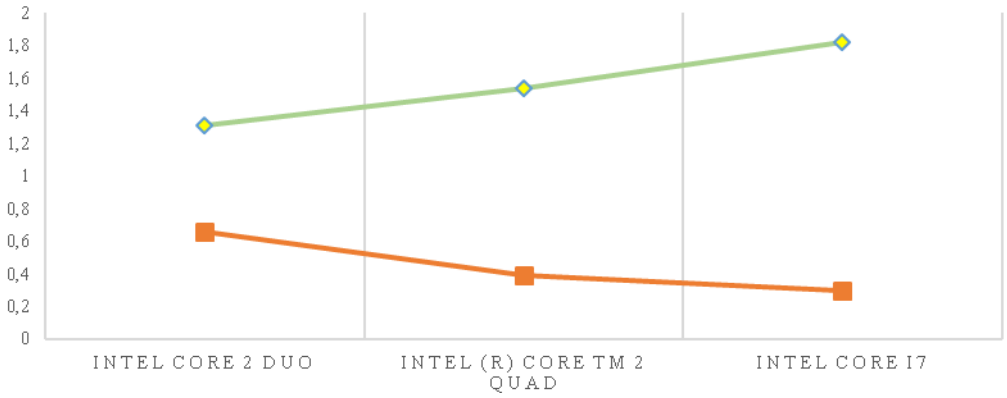

Figure 2. Acceleration and efficiency results taken for various hardware architectures. Efficiency in red and acceleration in green.

For the third scenario, the execution times of FuzzyPred in its parallel version were compared with the number of records in the database. The objective of this scenario is to know how much the execution time improves depending on the size of the databases. The configuration parameters are in Table 7 and the results of this scenario are shown in Table 8.

Table 7. Scenario 3 configuration parameters.

FuzzyPred Parameters	fuzzy logic operator=zadeh, connective=random, value scale=0-14, runs=50, iterations per run=10000, target=truth value, algorithm metaheuristic=random search, database=Quacke.
PC Characteristics	Intel Core i3 -2100 CPU 4 Gb de RAM

It is possible to argue that the FuzzyPred runtime is longer for Quacke because the database is much larger (contains more records), as shown in Table 8. In addition, the size of the data is a relevant factor in the algorithm runtime. It is important to note that the correspondence between data size and execution time is proportional as the time decreases according to the size of the database. Acceleration and efficiency metrics are inversely proportional measures, as shown in Figure 3.

Table 8. Execution times obtained for different databases.

Database	**Execution time**	**Database**	**Execution time**
Quacke	89 min	59 min	3589
Stulong	80 min	51 min	2458
Bolts	71 min	19 min	39

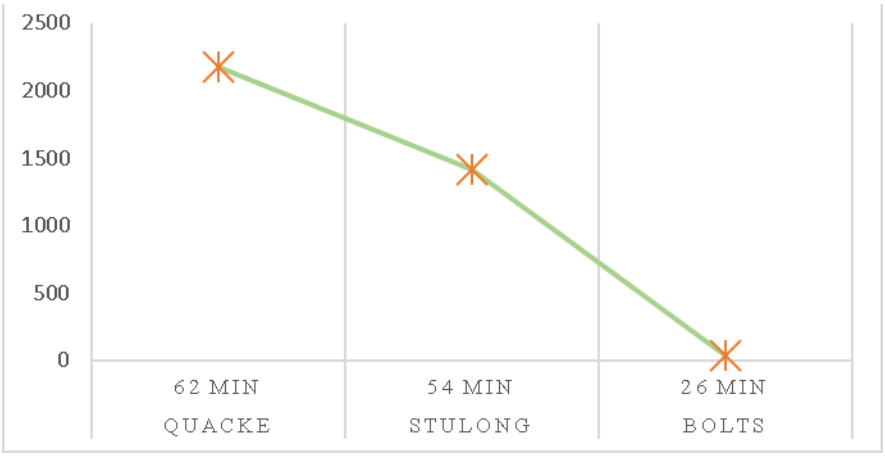

Figure 3. Results of parallel runtime versus multiple databases (number of records)

5. Conclusions

This study presents a data parallelism design implemented in the Java Parallel library. The proposed parallel design manages to reduce the runtime of the sequential version. The model is based on dividing the amount of data depending on the number of processors in the hardware architecture. The experimental results confirmed that the parallel version manages to reduce the sequential version by 10%. The experiments allow to verify that the results improve according to the hardware characteristics, in a proportional way and that the algorithm is faster in smaller databases. Other tests with larger databases and other types of hardware architectures are suggested.

References

[1] Chapman B, G. Jost and R Van der Pas. Using OpenMP: Portable Shared Memory Parallel Programming Scientific and Engineering Computation. The MIT Press.Massachusetts Institutte of Technology. ISBN 978-0- 262-53302-7. pp 349. 2008.
[2] Jain, Mugdha, and Chakradhar Verma. "Adapting k-means for Clustering in Big Data." International Journal of Computer Applications 101.1 (2014): 19-24.
[3] Ceruto T, O. Lapeira, A. Rosete and R. ESPÍN.Discovery of fuzzy predicates in database. Advances in Intelligent Systems Research (AISR Journal), vol. 51, No 1, pp. 45-54, ISSN 1951-6851, Atlantis Press, 2013.
[4] Hariri S, and M. Parashar.Tools and Enviroments for Parallel and Distributed Computing. John Wiley & Sons. ISBN 0-471-33288-7, pag 229, 2014.
[5] Fernandez A, S. Del Rio, V. Lopez, M. J. Del Jesus and F. Herrera. Big Data with Colud Computing:an insight on the computing enviroment, Map Reduce and programming frameworks. WIREs Data Mining and Knowledge Discovery.John Wiley and Sons, vol 4, pp 380-409, 2014.
[6] Viloria, A. "Commercial strategies providers pharmaceutical chains for logistics cost reduction." Indian Journal of Science and Technology 8, no. 1 (2016).
[7] Viloria, A., & Gaitan-Angulo, M. (2016). Statistical Adjustment Module Advanced Optimizer Planner and SAP Generated the Case of a Food Production Company. Indian Journal Of Science And Technology, 9(47). doi:10.17485/ijst/2016/v9i47/107371.
[8] Pas, R. An Overview of OpenMP 3.0. In., 2009.IWOMP. Tu Dresden (Alemania). Disponible en http://iwomp.zih.tu-dresden.de/downloads/2.Overwiew_OpenMP.pdf.

[9] N. Sapankevych y R. Sankar, "Time Series Prediction Using Support Vector Machines: A Survey", IEEE Computational Intelligence Magazine, vol. 4, núm. 2, pp. 24–38, may 2009.
[10] Reinders, J. Intel threading building blocks-outfitting C++ for multi-core processor parallelism. OReilly Media. ISBN 978-1449390860, pp 336, 2007.
[11] Kaminsky, A. The Parallel Java 2 Library Parallel Programming in 100 % Java. Rochester Institute of Technology, Department of Computer Science, Rochester, New York, EUA. 2015.
[12] F. Villada, N. Muñoz, y E. García, Aplicación de las Redes Neuronales al Pronóstico de Precios en Mercado de Valores, Información tecnológica, vol. 23, núm. 4, pp. 11–20. 2012.
[13] Venugopal K, K.G. Srinivasa and L. M. Patnaik. Soft Computing for Data Mining Applications. Springer Berlin Heidelberg: Springer-Verlag. ISBN 978-3-642-00192-5, pp 354, 2009.
[14] Brdar S., Culibrk D., Marinkovic B., Crnobarac J., Crnojevic V. Support Vector Machines with Features Contribution Analysis for Agricultural Yield Prediction, Second International Workshop on Sensing Technolo- gies in Agriculture, Forestry and Environment, 43-47, 2011
[15] Choudhury, A. and Jones, J. Crop yield prediction using time series models, Journal of Economics and Economic Education Research., 15, 53-68, 2014.
[16] R. Putha, L. Quadrifoglio, and E. Zechman. Comparing ant colony optimization and genetic algorithm approaches for solving traffic signal coordination under oversaturation conditions. Computer- Aided Civil and Infrastructure Engineering, 27(1), 14-28, 2012.
[17] D. Teodorović, and M. Dell'Orco. Mitigating traffic congestion: solving the ride-matching problem by bee colony optimization. Transportation Planning and Technology, 31(2), 135-152, 2008.
[18] A. L. Bazzan, and F. Klügl. A review on agent-based technology for traffic and transportation. The Knowledge Engineering Review, 29(3), 375-403, 2014.
[19] Amelec, V., & Alexander, P. (2015). Improvements in the automatic distribution process of finished product for pet food category in multinational company. Advanced Science Letters, 21(5), 1419-1421.
[20] Karatzoglou A., Smola A., Hornik K. and Zeileis A. kernlab - An S4 Package for Kernel Methods in R. Journal of Statistical Software, 11(9), 1-20, 2004