International Journal of Engineering & Technology, 7 (4) (2018) 4702-4705

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET
doi: 10.14419/ijet. v7i4.18058
Research paper



Performance evaluation of massive data standardization using multicore CPU and GPU

Saad Ahmed Dheyab [1] *, Dr. Buthainah Fahran Abed [2], Dr. Mohammed Najm Abdullah [3]

[1] College of Engineering, University of Information Technology and Communications, Ph.D. candidate in ICCI, Baghdad, Iraq.
[2] Smart Cities Dept., University of Information Technology and Communications, Baghdad, Iraq
[3] Computer Engineering Dept., University of Technology, Baghdad, Iraq
**Corresponding author E-mail: saad.theyab@uoitc.edu.iq*

Abstract

Standardization is one of the most important methods for the preprocessing phase in machine learning. It increases the quality of the results in terms of accuracy. Researchers have focused on the development of these preprocessing methods to suit the diversity of data generated from different sources. In this paper, three types of standardization methods (z score, min-max, log2) were applied to a massive dataset using three different preprocessing approaches (CPU single core, CPU multicore open MP, and GPU) and evaluated their performance. From the results, these approaches showed a faster GPU performance compared to the conventional CPU performance.

Keywords: *Standardization; GPU; Massive Data; Preprocessing.*

1. Introduction

Parallel computing is an approach towards accelerating many computationally intensive algorithms, Such as those that are used in image processing, simulations, sound and video applications, machine learning, data mining, and security devices [1]. Recently graphics Processing Units (GPUs) have been largely deployed as parallel computing engines [2]. Researchers are more interested in developing the use of the General Purpose Computing on Graphics Processing Unit (GP-GPU) programming model after the introduction of the Compute Unified Device Architecture (CUDA) framework by NVIDIA Corporation. Which the applications found in several fields like machine learning, data science, bioinformatics, and data mining [3].

As a contribution, this paper introduced the creation of the CUDA versions of the parallel standardization frameworks to be parallelly executed on single or multiple GPU cards. These CUDA versions can speed up the computational performance of the systems compared to the CPU. The performance of the novel algorithms was evaluated on a dataset comprised of several elements with 42 attributes and collectively compared.

The remaining part of this paper is presented as follows: Section 2 reviewing the previous related works, focusing on the pressing issues associated with massive datasets. Section 3 introducing a brief explanation of data standardization and its suitability in solving complex machine learning problems. Section 4 focusing on the important parallel computing approaches and the CUDA framework that enabled the GP-GPU programming model. Section 5 showing the details of the programming model proposed in this study. Section 6 presented the codes of the proposed algorithms, while section 7 discussed the results of the performance evaluations. The conclusions from the study are presented in the last section of the paper.

2. Related works

The interest in GPU computing has increased over the years as the need for more data processing using machine learning increases. Several articles on the implementation of CUDA in machine learning (ML) GPU frameworks and other algorithms are hereby, reviewed.

A study on the comparison between different ML approaches and a simple BSP-based model in the prediction of the execution time of GPU applications has been presented [4]. In this analysis, experiments were conducted using 9 different applications that can execute vector operations and 9 different NVIDIA GPUs made up of 6 Kepler and 3 Maxwell architectures.

A new brute force algorithm for the building of k-nearest neighbor graph has been described [5]. There are two parts of the proposed algorithm; the first part involves finding the distances between the input vectors, while the second part involves the selection of the k-neighbors for the testing sample. Furthermore, a new quick sort-based algorithm was implemented for a faster sorting of the distance pairs. The speed of the new algorithm was observed to increase with an increase in the k-variable.

The use an ML approach for the prediction of SpMV kernel performance on GPU algorithm has been proposed [6]. Two popular ML algorithms (Support Vector Regression (SVR) and Multilayer Perceptron neural network (MLP)) were used in the proposed concept. From the experimental results, the SVR was found to achieve a better accuracy on the two different GPUs (Fermi GTX 512 and Maxwell GTX 980 Ti) by presenting an average prediction error in the range of 7 - 14%.

In this work, three different ML preprocessing techniques were compared over CPU single core using c++, CPU multicore with open MP, and NVIDIA GPU using CUDA. These techniques were also compared over each hardware processing in terms of their performances.

Copyright © 2019 Saad Ahmed Dheyab et. al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

3. Data standardization

Data preprocessing plays a significant role in machine learning as it is a major step towards data standardization. This is a crucial step, especially when using parameters of different scales and units. For instance, some ML techniques utilize the Euclidean distance where all parameters are expected to have a similar to ensure a fair comparison. There are 3 known methods for data rescaling during standardization. These methods are the Z-score method, the Min-Max method, and the Log2 method. The scales of these methods are all numeric variables in the range [0-1]:

3.1. Z- score method

The Z-score or Zero mean normalization method involves the normalization of data based on their mean and standard deviation (SD) [7] using Formula 1:

$$d' = \frac{d - mean(P)}{std(p)} \quad (1)$$

Where Mean (p) = sum of P attribute values, and Std(P) = SD of all the P attribute values.

3.2. Min -max method

In the Min-Max method, the original data is linearly manipulated, with the values normalized within a certain range. To map the v-value of attribute A from a minA-maxA range to a new_minA, new_maxA range, the computation shown in Formula 2 is performed.

$$\frac{v - min_A}{max_A - min_A}(new_max_A - new_min_A) + new_min_A \quad (2)$$

Where v = new value in the desired range.
The advantage of this normalization method is in the annealing of all the values to a certain range [7].

3.3. Log2 method

To make a better decision during the data preprocessing phase and to reduce the computational overhead, it is necessary to convert the original dataset into a normal form. The continuous-valued features are discretized using a logarithm to the base 2 before casting the resulting value to the integer to avoid bias. For each continuous-valued z, 3 steps are involved [8].

$$\text{If } (z \geq 2)\ z = \int \log 2\ (z + 1) \quad (3)$$

4. Parallel computing

Parallel programming involves the use of several processors to simultaneously perform different aspects of the same program with the aim of reducing the total process execution time [9]. Parallel programming is mainly used for two main reasons; the first reason is to reduce the required time for troubleshooting and solving complex problems, and the second reason is that it is possible to take advantage of computing resources that may not be locally available or underutilized. It also helps to overcome memory-related challenges especially when the available memory on a single system may not be enough to solve a given problem. Furthermore, it helps to overcome the physical limits of miniaturization and speed which currently restricts the chance of constructing sequential computers. The next paragraph is focusing on parallel programming models (CUDA and OpenMP). The performance of these APIs is analyzed and compared in terms of speedup and runtime [10]:

4.1. GPU programming

The CPU consists of cores which have been optimized by using some serial processing, while GPU consist of several smaller cores designed for parallel performance, a CPU-GPU combination will be ideal. The CPU executes the serial parts of the code while the GPU executes the parallel parts. GPU computing involves the collective use of a graphics processing unit and a CPU to speed up general purpose applications in the science and engineering fields.
CUDA is a parallel programming model and a computing platform that enables a dramatic increase in the performance of a system using GPU. The concept of the language is that developers rely on the power of the GPU to perform massive operations in a faster way than using the CPU [9].

4.2. Open MP programming

The Open MP standard was developed and maintained by Open MP Architecture Review Board (ARB), a conglomerate of some big companies like SUN Microsystems, SGI, IBM, Intel, and others. Their API focuses on certain directives that support the development of parallel programs with shared memory through the implementation of an optimized and automatic set of threads. The use of OpenMP is useful in many aspects, as shown in its robust support for parallel programming, simplicity and slight changes in the code, ease of understanding, one support nested parallelism, use of directives and the possibility of dynamically adjusting the number of threads used [11].

5. Programming models

In this paper, three algorithms for standardization (z-score, min-max, and log2) were implemented using three different programming models described below:

5.1. CPU single core using c++

In this model, the algorithms were implemented in a traditional way and processed using a single processor core of core i7 7th CPU. They were serially executed on a massive data of millions of elements * and 42 attributes. The execution time of each algorithm was measured and compared to the parallel execution models.

5.2. CPU multicore using open MP

In this model, the algorithms were implemented in a parallel way and processed using 8 processor cores of core i7 7th CPU. They were parallelly executed using open MP directives on a massive data of millions of elements * and 42 attributes.

5.3. GPU many core using CUDA

In this model, the algorithms were implemented in a parallel way and processed using 768 processor cores of GPU GTX 1050ti. They were parallelly executed using CUDA on a massive data of millions of elements * and 42 attributes.
Table 1 showed the details of each programming model.

6. Reduction and algorithms

Reduction is one of the most prominent processes that can reduce execution time in parallel processing and used in finding the summation, maximum and minimum elements. Figure 1 shows how to find the maximum element in reduction parallel processing. In the results, the differences in the execution times using reduc-

tion of the massive data, specifically in the min-max algorithm, will be established.

Table 1: Pprogramming Models

	CPU single core	CPU multicore	GPU many core
Processor	Core i7 7th	Core i7 7th	GTX 1050 ti
Programming	C++	Open MP	Cuda
Cores no.	1	8	768
Execution	Serial	Parallel	Parallel

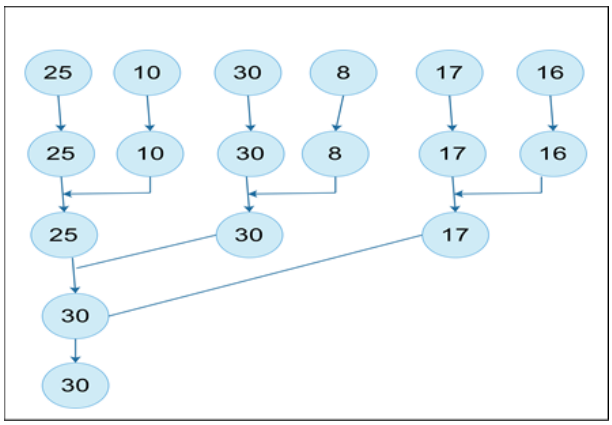


Fig. 1: Finding The Maximum Element Using Reduction Parallel Processing.

The next paragraph shows the details of all the algorithms implemented in this paper.

Algorithm 1: Z-score in single core CPU
Input: data (million*42)
Output: standardized data
Loading data sample from testing data;
For each attribute, do
For each sample, do
Compute mean;
Compute STD;
Compute z score value
End
End

Algorithm 2: Min-max in single core CPU
Input: data (million*42)
Output: standardized data
Loading data sample from testing data;
For each attribute, do
For each sample, do
Compute min;
Compute max;
Compute standardize value
End
End

Algorithm 3: Log2 in single core CPU
Input: data (million*42)
Output: standardized data
Loading data sample from testing data;
For each attribute, do
For each sample, do
Compute log2 (value+1)
End
End

Algorithm 4: Z-score in open MP
Input: data (million*42)
Output: standardized data
Loading data sample from testing data;
#pragma omp parallel for
For each attribute, do
For each sample, do
Compute mean;
Compute STD;
Compute z score value
End
End

Algorithm 5: Min-max in open MP
Input: data (million*42)
Output: standardized data
Loading data sample from testing data;
#pragma omp parallel for
For each attribute, do
For each sample, do
Compute min;
Compute max;
Compute standardize value
End
End

Algorithm 6: Log2 in open MP
Input: data (million*42)
Output: standardized data
Loading data sample from testing data;
#pragma omp parallel for
For each attribute, do
For each sample, do
Compute log2 (value+1)
End
End

Algorithm 7: Z-score in CUDA
Input: data (million*42)
Output: standardized data
Loading data sample from testing data;
#does in parallel with reduction
For each attribute, do
For each sample, do
Compute mean;
Compute STD;
Compute z score value
End
End

Algorithm 8: Min-max in CUDA
Input: data (million*42)
Output: standardized data
Loading data sample from testing data;
#do in parallel with reduction
For each attribute, do
For each sample, do
Compute min;
Compute max;
Compute standardize value
End
End

Algorithm 9: Log2 in CUDA
Input: data (million*42)
Output: standardized data
Loading data sample from testing data;
#Do in parallel with reduction
For each attribute, do
For each sample, do
Compute log2 (value+1)
End
End

7. Reduction and algorithms

The execution time of each algorithm was measured and compared to find the best method that can speed up the performance of the algorithms in processing massive data. Table 2 described the details of the observed execution times of the algorithms.

Table 2: Execution Times of the Algorithms (S)

	CPU single core	CPU multicore	GPU many core
Min-max	0.85	0.32	0.13
Z-score	3.1	0.61	0.15

	CPU single core	CPU multicore	GPU many core
Log2	7.4	1.35	0.39

Figure 2 showed the differences in the execution time of each algorithm. Log2 algorithm using GPU was 18 times faster than CPU single core and 3 times faster than CPU multicore open MP. Z-score algorithm using GPU was 20 times faster than CPU and 4 times faster than open MP. Finally, Min-max algorithm using GPU was 6 times faster than CPU and 2.4 times faster than open MP. The results showed the GPU to have a better performance in massive data processing compared to the other processing approaches.

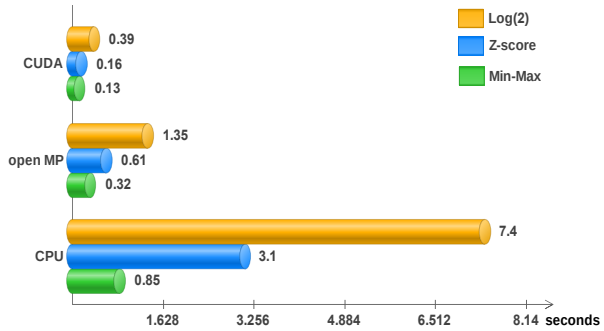


Fig. 2: Comparision of the Execution Time (S) of the Algorithms.

8. Conclusion

The major contribution of this study is the development of the CUDA versions of parallel standardization algorithms which can be executed on single or several GPU cards in parallel. The use of GPU accelerates the processing of massive data which may not be efficiently processed using the traditional methods. This paper showed the effect of using reduction methods in parallel data processing. The CPU comprises 8 cores and slower in data processing compared to GPU processor with 768 Cores. A dataset composed of millions of elements and 42 attributes was processed using GPU, GTX, 1050, and Core i7 processor in less than a second.

References

[1] Masek, Jan & Burget, Radim & Povoda, Lukas & Kishore Dutta, Malay. (2016). Multi–GPU Implementation of Machine Learning Algorithm using CUDA and OpenCL. International Journal of Advances in Telecommunications, Electrotechnics, Signals, and Systems. 5. https://doi.org/10.11601/ijates.v5i2.142.

[2] Kirk, D. B., & Hwu, W. W. (2013). Programming massively parallel processors (2nd edition). Waltham, MA: Elsevier Inc.

[3] Abdul Hay Bin Sulaiman, Muhamad & Suliman, Azizah & Ahmad, Abdul. (2014). Measuring GPU-accelerated parallel SVM performance using large datasets for multi-class machine learning problem. 299-302. 10.1109/ICIMU.2014.7066648.

[4] M. Amaris, D. Cordeiro, A. Goldman, and R. Y. Camargo, "A simple bsp-based model to predict execution time in GPU applications," in High-Performance Computing (HiPC), 2015 IEEE 22nd International Conference on, December 2015, pp. 285–294. https://doi.org/10.1109/HiPC.2015.34.

[5] I. Komarov, A. Dashti, R. D Souza,"Fast k-NNG construction with GPU based quick multi-select",2013.

[6] Benatia, Akrem & Ji, Weixing & Wang, Yizhuo & Shi, Feng. (2016). Machine Learning Approach for the Predicting Performance of SpMV on GPU. 894-901. https://doi.org/10.1109/ICPADS.2016.0120.

[7] Saranya, C & Manikandan, G. (2013). A study on normalization techniques for privacy-preserving data mining. 5. 2701-2704.

[8] Haddadpajouh, Hamed & Dastghaibyfard, Gholamhossein & Hashemi, Sattar. (2015). Two-tier network anomaly detection model: a machine learning approach. Journal of Intelligent Information Systems. https://doi.org/10.1007/s10844-015-0388-x.

[9] P. S. Pacheco. An Introduction to Parallel Programming. University of San Francisco, 2013.

[10] D. B. Kirk. Programming Massively Parallel Processors, Second Edition: A Hands-on Approach. 2013.

[11] Ms. Ashwini M. Bhugul, (2017), "Parallel Computing using OpenMP", International Journal of Computer Science and Mobile Computing, vol 6, issue 2, p90-94.