Parallelization of Linear Regression

Sami Rodrigue and Suansh Sharma

**1: Introduction**

Improvements in the computational power and in-memory storage of personal computers along with the ease of scaling up thanks to cloud computing services have opened data analytics to a broader audience. This in turn played an important role in building the excitement around Big Data and machine learning. Today, anyone can rapidly prototype any model they choose from a popular machine learning library without needing to worrying about the efficiency of their implementation. However, to generate production quality code inefficiencies cannot be easily overlooked. One easy way to improve the efficiency is using parallelization.

Linear regression is one of the most commonly used models in data analytics because it is easy to implement and interpret. Additionally, it has been studied extensively, hence its strengths and weaknesses are well-known. To build a linear regression, the user needs to determine which variable they want to model, the target variable, and the variables they want to use, independent variables, to explain this chosen variable. Once this selection is done, they can reach conclusions about the target variable assuming they keep linear regression’s limitations in mind.

Due to its wide use, implementing a scalable linear regression is an important addition to a Data Scientist’s arsenal. The aim of this paper is to show the benefits of parallelizing linear regression and showcase different approaches to achieving this task.

The rest of the paper is organized as following; Section 2 presents a basic introduction to linear regression and how it is solved. Section 3 summarizes the literature related to parallelizing linear regression. Section 4 goes over the methods we have used to parallelize linear regression. Section 5 presents our simulation results and Section 6 concludes the report.

**2: Background**

In linear regression [1][2] we can define a model by where y is the target variable, β0 is the intercept, x is the independent variable used to interpolate the target variable which has the regression coefficient β1, and ε is the error term which has a Normal distribution with a mean of 0 and standard deviation of σ2. This model can be generalized to multiple features by simply adding more independent variables to the right hand-side of the formula.

If we were to write the model in matrix notation (without the error term) it would be written as following:

To solve the linear regression model, we need to employ a method called least-squares. This method can be represented as . In essence, we are trying to find the projection of y onto X matrix’s column space that minimizes the distance between the y vector and the X’s column space [3-4].

**3: Related Literature**

Cheng-Tao et al. [5] has used the MapReduce paradigm to parallelize linear regression, naïve Bayes, k-means, Neural Network and other commonly used tools in the machine learning community. This paper can be replicated by using Spark’s MLlib which includes many widely used machine learning algorithms. Xu [8] parallelizes multiple linear regression by partitioning the features and the target variable into smaller subsets. Then Xu measures how linear regression fits the subset by calculating the least squares estimate for each cluster.

Parallelization of linear regression has also been analyzed for more specialized cases such as when it is built using sparse datasets [6]. Van Heeswijk et al. [7] has parallelized linear regression using an ensemble of Extreme Learning Machines (ELM) that are trained on GPUs. To best of our knowledge, the literature for parallelization of linear regression generally uses approximations or gradient descent. This paper will attempt to improve the computation time for the closed-form solution of linear regressions.

**4: Implementation**

**4.1: Task Analysis**

Calculating the least-squares of a dataset is overall a static process as a result we can use static mapping for the associated tasks. Our reasons are as following:

* The input has a statics size and does not change during run time.
* The tasks sizes are uniform because we have concentrated our efforts on dense matrix.
* The tasks have static interactions and can be mapped a-priori.

**4.2: Method**

As introduced in the Background section of this paper, the least-squares method can be easily parallelized by concurrently calculating the segments of the underlying matrix. In this paper we used two methods to parallelize the least squares method: Parallelizing the matrix multiplication and Parallelizing by computing the model on separate clusters. In the first method we are using parallelized matrix multiplication to compute the regression parameters and in second method we are dividing the data set into clusters and computing them in parallel. Both of these methods are discussed in detail in the subsequent sections.

**4.2.1: Method – 1: Parallelizing the Matrix Multiplication**

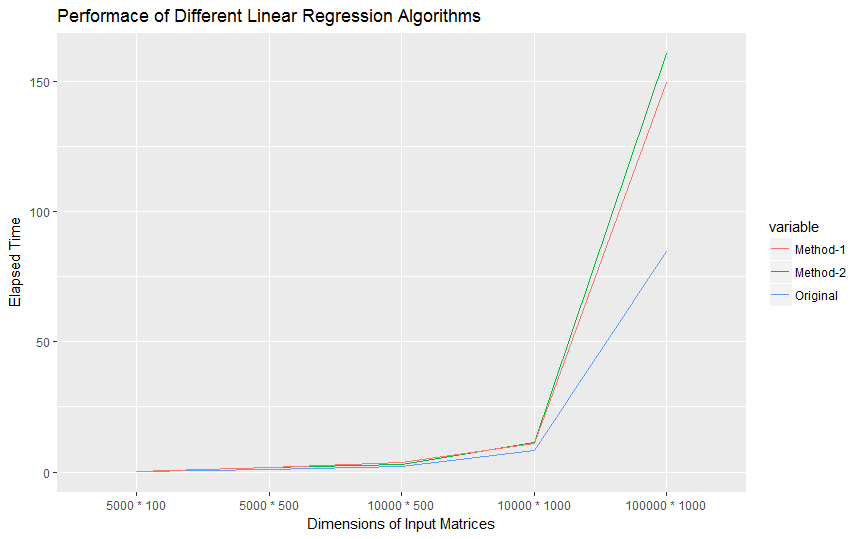
In this method we have decided to compute the regression parameters by parallelizing the matrix multiplication. We are using row-wise distribution of our matrix to parallelize the matrix multiplication. In row-wise distribution, we split our matrix row-wise into K subsets where K is the number of cores in the machine and assign each cluster a subset of the matrix. After splitting, we multiply each subset of the first matrix with the second matrix in the respective clusters. The outcome from each cluster is finally combined by rows to get the final matrix. In this method we are parallelizing all the matrix multiplications involved in least square method the same we just discussed. This method should be more efficient than the sequential multiplication when we are using a large matrix because of the overheads involved in feeding the data to clusters and reading the output from the clusters.

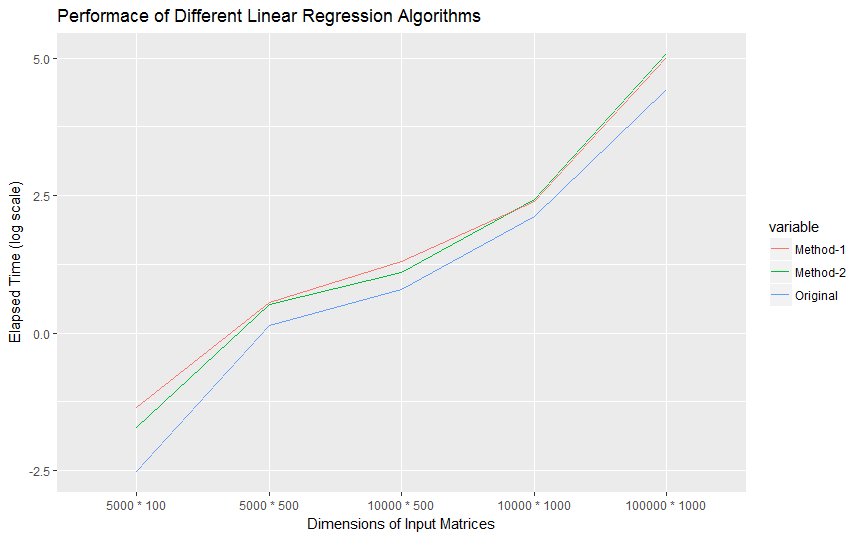
**4.2.1: Method – 2: Parallelizing by Computing on Separate Clusters**

We have also decided to divide our dataset in the K subsets where K is the number of cores in the machine. We have calculated K models in parallel and then took the average of the calculated coefficients. This method, like methods that are not closed-form, approximations of the actual model’s coefficients and it needs the subsets to have at least as many rows as their columns. Else, we cannot solve the linear system because we would have more unknowns then the number of equations. To solve this issue, we would need to utilize fewer cores then available. Additionally, least squares method is highly influenced by outliers and this methodology exacerbates the issue. If the dataset is divided in a way such that outliers are collected in a single core, then the resulting weights will highly influence the approximated coefficients.

**5: Results**

We compared our parallelized methods with the original sequential multivariate linear regression. We are using 64-bit Intel i7-6600U CPU at 2.60 GHz with 4 cores with 8 GB of RAM. We compared elapsed time for randomly created matrices of different sizes. We found that our parallelized algorithms do not outperform the sequential method. We are not seeing any improvement in both of the algorithms even for large input matrices over the sequential execution. The charts below show the comparison of the elapsed times of all the three methods for different input dimensions of the matrices.





We believe the reason for lack of improvement is our methods have introduced high communication costs between the clusters, causing the proposed methods to be slower. In our parallelized calculations, we pass the data between clusters back and forth multiple times. We also believe the message passing between the clusters to compile the results from all the clusters to get a final output is also adding a significant computation time. Additionally, we have observed that when the solve function is running, multiple CPUs are being utilized. This leads us to believe that the solve function, which uses LAPACK in the background, might already be parallelized by using a package like PLASMA or by using multithreaded programming.

**6: Summary**

We have examined the parallelization of linear regression by using two different methods. We parallelized linear regression by parallelizing its matrix multiplications and by parallel solving different subsets of the provided dataset. We found that both our algorithms do not perform better than the sequential method because of the overhead costs we have introduced and the efficiencies available in R’s solve function. This explains why gradient descent is widely used for larger datasets. Hogwild is one such technique widely used for the parallelization of stochastic gradient descent.

**References**

[1] Dalgaard, Peter. “Chapter 6: Regression and Correlation.” Introductory Statistics with R, Lightning Source UK Ltd., 2011.

[2] Dalgaard, Peter. “Chapter 11: Multiple Regression.” Introductory Statistics with R, Lightning Source UK Ltd., 2011.

[3] Strang, Gilbert. “Chapter 3.2: Cosines and Projections onto Lines.” Linear Algebra and Its Applications Fourth Edition, Thomson, Brooks/Cole, 2006.

[4] Chamberlain, Ph.D. Andrew. “The Linear Algebra View of Least-Squares Regression.” Medium, Medium, 10 Dec. 2016, medium.com/@andrew.chamberlain/the-linear-algebra-view-of-least-squares-regression-f67044b7f39b.

[5] Chu, Cheng-Tao, et al. "Map-Reduce for Machine Learning on Multicore." Advances in neural information processing systems. 2007.

[6] Mateos, Gonzalo, Juan Andrés Bazerque, and Georgios B. Giannakis. "Distributed Sparse Linear Regression." IEEE Transactions on Signal Processing 58.10 (2010): 5262-5276.

[7] Van Heeswijk, Mark, et al. "GPU-Accelerated and Parallelized ELM Ensembles for Large-Scale Regression." Neurocomputing 74.16 (2011): 2430-2437.

[8] Mingxian, Xu, John J. Miller, and Edward J. Wegman. "Parallelizing multiple linear regression for speed and redundancy: an empirical study∗." Journal of Statistical Computation and Simulation 39.4 (1991): 205-214.