Parallel methods for linear systems solution in extreme learning machines: an overview

Parallel methods for linear systems solution in extreme learning machines: an overview

E Gelvez-Almeida[1,2], Y Baldera-Moreno[1], Y Huérfano[3], M Vera[2], M Mora[1], and R Barrientos[1]

[1] Laboratorio de Investigaciones Tecnologicas en Reconocimiento de Patrones, Universidad Católica del Maule, Talca, Chile
[2] Facultad de Ciencias Básicas y Biomédicas, Universidad Simón Bolívar, San José de Cúcuta, Colombia
[3] Grupo de Investigación en Procesamiento Computacional de Datos, Universidad de Los Andes, San Cristóbal, Venezuela

E-mail: marcomoracofre@gmail.com

Abstract. This paper aims to present an updated review of parallel algorithms for solving square and rectangular single and double precision matrix linear systems using multi-core central processing units and graphic processing units. A brief description of the methods for the solution of linear systems based on operations, factorization and iterations was made. The methodology implemented, in this article, is a documentary and it was based on the review of about 17 papers reported in the literature during the last five years (2016-2020). The disclosed findings demonstrate the potential of parallelism to significantly decrease extreme learning machines training times for problems with large amounts of data given the calculation of the Moore Penrose pseudo inverse. The implementation of parallel algorithms in the calculation of the pseudo-inverse will allow to contribute significantly in the applications of diversifying areas, since it can accelerate the training time of the extreme learning machines with optimal results.

1. Introduction

The multilayer perceptron (MLP) [1] and support vector machines (SVM) [2] can be considered the most widely used in machine learning context due to their great ability to generalize in classification and regression problems. On the other hand, extreme learning machines (ELM) have aroused great interest because they achieve similar levels of performance as MLP and SVM but with substantially shorter training times. In ELM's training algorithm, the hidden layer weights and biases are set randomly and the output layer weights and biases are estimated by solving overdetermined system of linear equations (SLE), typically using the Moore-Penrose generalize inverse matrix [3].

Parallel programming is based on breaking a problem down into smaller, independent parts or processes; the independent processes are executed simultaneously by multiple processors that communicate with each other through memory, considering different approaches [4]. This allows code to be executed faster (acceleration) [4] compared with CPU serial execution time.

Several works have now been reported in which efforts were made to implement ELM using parallel techniques such as MapReduce and GPU implementation [5]. Subsequently, in [6] the main drawbacks of ELMs are presented, such as the difficulty to determine the hidden layers

structure, prediction of instability and unbalanced data distributions; likewise, in [7] a review on monolayer (SLFN) and multilayer (ML-ELM) extreme learning machines, is developed.

The aim of this work is to present an updated review of parallel algorithms for the solution of SLE, which can be applied for the computation of the Moore-Penrose generalized inverse matrix during the ELM training.

2. Extreme learning machine foundations

This section presents the main foundations to define the type of neural network (NN) addressed in this review.

2.1. Moore-Penrose generalized inverse matrix

The solution of an overdetermined SLE $Ax = y$, where $A \in \mathbb{R}^{m\times n}$ with $m > n$, can be simplified using the Moore-Penrose generalized inverse matrix $A^\dagger \in \mathbb{R}^{n\times m}$ [8], which is unique and satisfies all four conditions: $AA^\dagger A = A$, $A^\dagger AA^\dagger = A^\dagger$, $A^\dagger A = (A^\dagger A)^T$, and $AA^\dagger = (AA^\dagger)^T$.

2.2. Extreme learning machine

Let N be a training set $\{(x_i, t_i)|x_i \in \mathbb{R}^n, t_i \in \mathbb{R}^m\}$ with $i = 1, \ldots, N$. Equation (1) presents the learning algorithm for SLFNs with an activating function $g(s) : \mathbb{R}^n \to \mathbb{R}^m$ and L hidden layer neurons [9].

$$\sum_{i=1}^{L} \beta_i g(w_i x_j + b_i) = t_j, \qquad j = 1, \ldots, N, \tag{1}$$

where β_i are the output weights, w_i and b_i are the ith weight and bias of the input layer, respectively. The results of Equation (1) can be written as $H\beta = T$, where the matrix H is called the hidden layer output matrix of the NN [10]. The ith column of H is the ith output vector of the hidden neuron with respect to the input $(x_1, \ldots, x_N)$. In [11] the steps to calculate the output weights $\beta = H^\dagger T$ are presented, where $H^\dagger$ is the Moore-Penrose generalized inverse matrix of H.

Several variants of ELM have been proposed in the literature with the aim of achieving better results, such as ML-ELM [12]. In that sense, in [13] a ML-ELM architecture is presented using an ELM-based AutoEncoders (AE) approach (ELM-AE). The ML-ELM stacks ELM-AE to create a multilayer neural network [13].

3. Methodology

In this work, the comprehensive interpretive paradigm was used, employing the revision method through the content analysis technique. For the review, articles related to the object of study were selected and the relevant information was synthesized using a comparative technique, allowing the identification and analysis of both the foundations of the methods for solving systems of linear equations (MSSLE) and the formally reported antecedents.

For the analysis, the following methodological stages were developed: (i) Identification of the theoretical foundations related to MSSLE in their classic and parallel versions, (ii) Addressing the antecedents through critical review of units of analysis (papers and proceedings), in different specialized databases such as Springer, Elsevier and IEEE Xplore, academic databases and high impact journals, and (iii) classification of 17 documents according to the scope, year and context of the research, from the 2016 to 2020.

In the search, the types of documents discriminated were classified as follows: 6 documentary review papers, 17 articles that present proposals for models and strategies for the parallelization of MSSLE (4 in conferences, 1 in symposium, 12 in specialized journals).

4. Parallel methods for solving linear systems

This section briefly presents the methods based on operations, matrix factorization and iterative for the solution of SLE. For each method, a review of the most recent works that implement parallel algorithms is presented.

4.1. Methods based on operations

One of the main methods based on operations is Gaussian elimination (GE), which converts the equation $Ax = b$ into $Ux = y$, where U is a superior triangular matrix and y is a new vector of values of equation [8]. Table 1 presents an overview of more recent articles on feature-based parallel methods.

Table 1 presents an approach based on partitioning the matrices into blocks and the parallel implementation of Gaussian elimination. For this, it will be used in shared memory architectures and physical infrastructure to raise the quality of cryptographic systems through the application-specific integrated circuit. Likewise, a scheme of Multilevel hybrid parallel calculation for magnetotelluric occam inversion based on MPI + OpenMP + CUDA libraries, which proved to be feasible and efficient. It was evident that in most of the references, the scalability of the multicore CPU system architecture exhibits better performance than GPUs.

Table 1. Brief description about papers related to paralelized methods based on operations.

References	Applications	Data	Parallel architecture	Results
Yi H B *et al.* 2018 [14]	Gaussian Elimination in finite fields	In $GF(2^8)$ takes 739 clock cycles and around 2000 gate	TSMC-0.18$^{\mu}$m standard cell CMOS ASIC	Design is suitable for MQ cryptographic system implementations
Pan V Y, Zhao L 2017 [15]	Gaussian elimination with no pivoting	$N \times N$ matrix with $N = 256$ to $N = 4096$	Intel Core 2 2.50 GHz processor and 4G memory running Windows 7	Its efficiency is confirmed with numerical tests
Abouelfarag A A *et al.* 2016 [16]	Multicore architecture and GPU architecture using GE problem	$N \times N$ matrix with $N = 512$ to $N = 8192$	Two Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60 GHz. GeForce 8800 Ultra	Multicore performance outperforms GPU when sockets are increased
Liu Y *et al.* 2016 [17]	Parallel Gaussian Elimination	Parameters of Inversion Model $N_r = 21, 41, N_m = 889, 1549, NE(y \times z) = 103 \times 35, 193 \times 35$ nfre= $20, 36$	An Intel Core i5-3470 CPU and an GTX 680 GPU GDDR5 device memory	In the experiments carried out, the acceleration ratio is up to 23.49
Dumas J G *et al.* 2016 [18]	Parallelization of subcubic GE on shared memory architectures	$N \times N$ matrix with $N = 2000$ to $N = 30000$	32 cores Intel Xeon E5-4620 2.2 GHz with L3 cache	Building blocks combined in recursive mosaic algorithm give high computing efficiency

4.2. Methods based on matrix factorization

Among the effective algorithms for calculating the pseudoinverse matrix are the methods based on singular value decomposition (SVD), Cholesky factorization, QR factorization, tensor product and the conjugate process of Gram-Schmidt [3]. The SVD of a matrix $H \in \mathbb{R}^{m \times n}$ is defined by $H = U\Sigma V^T$, where U and V are unit matrices of size $m \times m$ and $n \times n$ respectively and Σ is a diagonal matrix of size $m \times n$ with diagonal elements $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ [8]. The pseudoinverse of H is given by $H^{\dagger} = V\Sigma^{\dagger}U^T$ [3, 8].

On the other hand, the Cholesky factorization is admitted for all symmetric and positive definite $A \in \mathbb{R}^{n\times n}$ matrix, for which there is a superior triangular matrix L of dimensions $n \times n$ such that $A = LL^T$ [8]. In [3] the algorithm for calculating the pseudoinverse using Cholesky factorization is presented. Regarding the QR factorization, if $A \in \mathbb{R}^{m\times n}$ has linearly independent columns, then A can be factored in the form $A = QR$, where Q is a matrix with orthonormal columns and R a superior triangular matrix [8]. For the pseudoinverse calculation, in [3, 19] algorithms implementing QR factorization are presented. Table 2 presents an overview of more recent articles on parallel methods based on matrix factorization.

The parallel matrix factoring algorithms for heterogeneous architectures connecting processors with different characteristics developed in Table 2 present notable advantages compared to traditional methods. These methods achieve high performance, adequate precision and scalability in the solution of large symmetric matrix data using MPI and CUDA.

Table 2. Analysis about parallel factorization methods using papers reported in literature.

References	Applications	Data	Parallel architecture	Results
Lu Y *et al.* 2020 [20]	SVD factorization and video processing	$M \times N$ double precision matrix with $M = 10$ and $N = 5 \times 10^1$ to $N = 5\times10^5$, and a high definition traffic video	Two Intel 8-core Xeon Silver 4110 CPUs and two Tesla V100 (Volta) GPUs	Some methods acceleration up to 5.2 times
Tomás A *et al.* 2019 [21]	SVD factorization	$N \times N$ single and double precision matrix from $N = 5 \times 10^3$ to $N = 3 \times 10^4$	Tesla P100 with Intel 8+8-core Xeon E5-2620 v4, GeForce Titan X with Intel 4-core Core i7-3770K, and Tesla K20c with Intel 6-core i7-3930K	Significantly decreases parallel execution time
Wu R 2019 [22]	Cholesky Factorization	$N \times N$ double precision matrix with $N = 10000$ and $N = 20000$	Two 8-core Intel Xeon E5-2620 v4 CPUs	Parallel execution times progressively decrease
Wu R 2018 [23]	Cholesky Factorization	$N \times N$ double precision matrix from $3N$ to $7N$ ($N = 16384$) in different system configurations	Two 8-core Intel Xeon E5-2640 CPUs and eight Tesla M2090 GPUs	Good performance on scalability, communication cost, and load balancing
Zhang S *et al.* 2020 [19]	QR factorizatión on neural engines	Large-scale rectangular matrix of different sizes	Tesla V100 (PCIe version) GPU	Cost efficiency using neural motors to accelerate linear algebra operations
Tapia-Romero M *et al.* 2020 [24]	QR factorization	$N \times N$ single and double precision matrix from $5N$ to $20N$ ($N = 1\times10^3$) in different system configurations	Two 6-core Intel Xeon X5675, and two Tesla K20X, one Tesla C2070, and one GeForce GTX 460	The combination of MPI and CUDA is feasible

4.3. Iterative methods

Iterative methods are used in large spaced systems and are efficient in both storage and computational time; the most used iterative methods are: Jacobi, Gauss-Seidel and successive over relaxation (SOR). An iterative method by which the SLE $Ax = b$ is solved begins with an initial approximation $x^{(0)}$ to the solution x and generates a sequence of vectors $\{x^{(k)}\}_{k=0}^{\infty}$ that

converges to x. Iterative methods bring along with them a process that converts the system $Ax = b$ into another equivalent of the form $x = Tx+c$ for some fixed matrix T and a vector c [8]. After selecting the initial vector $x^{(0)}$, the sequence of the vectors of the approximate solution is generated by calculating $x^{(k)} = Tx^{(k-1)} + c$, for $k = 1, 2, 3, ..., n$. To solve a system $Ax = b$, where $A \in R^{n\times n}$ and $x, b \in R^n$, the Jacobi method performs calculations using matrix notation $x^{(k)} = D^{-1}(L+U)x^{(k-1)} + D^{-1}b$ where, $A = D-(L+U)$ is a division of A into its diagonal D, upper triangular L and lower triangular U, respectively [8].

On the other hand, the Gauss-Seidel [8] method can be implemented using a single iteration vector, which is important for systems of large linear equations. The Gauss-Seidel method, in which there is no reason to conserve $x^{(k)}$ to calculate complete $x^{(k+1)}$, is called a SOR method [8]. These methods have a high degree of parallelism. Table 3 below presents a summary of the most recent articles on parallel algorithms based on iterative methods.

The references reported in Table 3 show that parallelized iterative methods allow to solve structured problems in GPU units efficiently. Some of these methods include OpenMP dynamic programming and MPI communication which have improved execution time compared to other methods. Additionally, as the GPU is a highly parallel structure with thousands of cores, the GPU performance for iterative methods exhibited more efficient performance than the CPU.

Table 3. Summary related to paralelized methods based on iterative methods.

References	Applications	Data	Parallel architecture	Results
Islam M S *et al.* 2020 [25]	Jacobi method. Structured problems	SLE that arise from the discretization of the Poisson equation in 1D and 2D	GPU parallel deployment using shared memory	Optimal performance of the shared memory algorithm
Yang X *et al.* 2018 [26]	Jacobi Stencil Algorithm	Information analysis center of the Sanjiangyuan region as a test bed	Cluster with 48 compute nodes, and each node has two CPUs and 24 cores	The execution time is significantly reduced compared to the version of a singlethread
Aslam M *et al.* 2020 [27]	Nine Variants of the Jacobi Method	Sparse matrices with double precision arithmetic	Intel Core-i7 8750h CPU and two NVIDIA GPUs	GPUs achieve an acceleration of at least 46 times with respect to CPUs
Naik T U *et al.* 2017 [28]	Gauss-Seidel method. Symmetric matrices	Size matrices: 500×500, 2000×2000, 3000×3000, 4000×4000, 5000×5000	CUDA on GPU	GPU performance is faster than CPU
Wu Z *et al.* 2017 [29]	Gauss-Seidel method. Signal detection algorithm	Massive 128×8 MIMO system on a Xilinx Virtex-7 XC7VX690T FPGA	PGS detector for a massive 128×8 MIMO system on a Xilinx Virtex-7 FPGA XC7VX690T	PGS provides higher performance than conventional GS detector
Huang G H *et al.* 2020 [30]	SOR method. Linear equations of discontinuous deformation analysis	Equations in 3D spheres. Application example with 10,000 spheres and 200,000 calculation steps	OpenMP-based serial and parallel computing performed on a 16-core PC	SOR is approximately six times faster than other methods for serial computing

5. Results

The ELM is a neural model that is implemented in many applications with a fast learning speed and good generalizability. Currently, the high volume of data presents in matrices with large

dimensions reuires learning time increase for calculating H pseudoinverse. One of the most viable options to counteract this increase is to parallelize the algorithms of the pseudoinverse calculation with multicore CPUs and GPUs. Parallelization focuses on solving overdetermined SLE with methods based on operations, factorizations and iterations.

The review of the three methods in the literature has shown a significant decrease in parallel times implementing different architectures in distinctive ways. To conduct the experiments, the authors have used a variety of data such as images and videos. In the literature, it was reported papers with square and rectangular single and double precision matrix, with sizes ranging from 10 to 5×10^5 rows and/or columns, having very good performance with large matrices. Some of the limitations that parallel models present have to do with hardware memory.

6. Conclusions

An updated review of parallel algorithms for solving SLE has been presented in this paper. Solving SLE is a complex problem since, when working with large volumes of data, the size of an array associated with the system increases, which in turn increases the computation time of the pseudo-inverse. It is important to find an optimal solution, using high-performance computing. That is why the methods based on operations, matrix factorizations and iterations were addressed in their parallel versions. The parallel method based on Gaussian elimination showed optimal performance when using multicore CPU architecture compared to GPU for solving large-dimensional SLE.

On the other hand, it is evidenced that parallel algorithms significantly accelerate the factoring process of a large matrix, and at the same time have good overall performance (scalability, load balancing, and communication cost). The iterative methods addressed were Jacobi, Gauss-Seidel and SOR. It was concluded that these methods are suitable for parallel architectures since they allow to solve problems of SLE in GPU units efficiently. These methods outperform performance of other methods reported in the literature.

Acknowledgments

The authors of the paper thank to research project FONDECYT regular 2020 Nº 1200810 “Very large fingerprint classification based on a fast and distributed extreme learning machine”, Ministerio de Ciencia, Tecnología, Conocimiento e Innovación, Gobierno de Chile. National Agency for Research and Development (ANID), scholarship program, Beca Doctorado Nacional, Chile 2020.

References

[1] Hinton G E and Salakhutdinov R R 2006 Reducing the dimensionality of data with neural networks *Science* **313(5786)** 504–507
[2] Lin C F and Wang S D 2002 Fuzzy support vector machines *IEEE Transactions on Neural Networks* **13(2)** 464–471
[3] Lu S, Wang X, Zhang G and Zhou X 2015 Effective algorithms of the moore-penrose inverse matrices for extreme learning machine *Intelligent Data Analysis* **19(4)** 743–760
[4] Rauber T and Rünger G 2013 Performance analysis of parallel programs *Parallel Programming* (Berlin: Springer) pp 169–226
[5] He Q, Shang T, Zhuang F and Shi Z 2013 Parallel extreme learning machine for regression based on mapreduce *Neurocomputing* **102** 52–58
[6] Alaba P A, Popoola S I, Olatomiwa L, Akanle M B, Ohunakin O S, Adetiba E, Alex O D, Atayero A A and Daud W M A W 2019 Towards a more efficient and cost-sensitive extreme learning machine: A state-of-the-art review of recent trend *Neurocomputing* **350** 70–90
[7] Parkavi R M, Shanthi M and Bhuvaneshwari M C 2017 Recent trends in elm and mlelm : A review *Advances in Science, Technology and Engineering Systems Journal* **2(1)** 69–75
[8] Lyche T 2020 *Numerical Linear Algebra and Matrix Factorizations* (Oslo: Springer)
[9] Hornik K, Stinchcombe M, White H *et al.* 1989 Multilayer feedforward networks are universal approximators. *Neural networks* **2(5)** 359–366

[10] Huang G B, Zhu Q Y and Siew C K 2004 Extreme learning machine: a new learning scheme of feedforward neural networks *IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)* (Budapest: IEEE)
[11] Salazar E, Mora M, Vásquez A and Gelvez E 2020 Conditioning of extreme learning machine for noisy data using heuristic optimization *Journal of Physics: Conference Series* **1514** 012007:1
[12] Tang J, Deng C, Huang G 2016 Extreme learning machine for multilayer perceptron *IEEE Transactions on Neural Networks and Learning Systems* **27(4)** 809–821
[13] Kasun L, Zhou H, Huang G and Chi M 2013 Representational learning with elms for big data. intell. syst *IEEE Intelligent Systems* **28(6)** 31–34
[14] Yi H B, Nie Z and Li B 2018 Efficient implementations of gaussian elimination in finite fields on asics for mq cryptographic systems *Journal of Discrete Mathematical Sciences and Cryptography* **21(3)** 797–802
[15] Pan V Y and Zhao L 2017 Numerically safe gaussian elimination with no pivoting *Linear Algebra and its Applications* **527** 349–383
[16] Abouelfarag A A, Nouh N M, ElShenawy M 2016 Scalable parallel approach for dense linear algebra *International Conference on High Performance Computing & Simulation (HPCS)* (Innsbruck: IEEE)
[17] Liu Y, Xiong R and Xiao Y 2016 A MPI+ OpenMP+ CUDA hybrid parallel scheme for MT occam inversion *International Journal of Grid and Distributed Computing* **9(9)** 67–82
[18] Dumas J G, Gautier T, Pernet C, Roch J L, Sultan Z 2016 Recursion based parallelization of exact dense linear algebra routines for gaussian elimination *Parallel Computing* **57** 235–249
[19] Zhang S, Baharlouei E and Wu P 2020 High accuracy matrix computations on neural engines: A study of qr factorization and its applications *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing* (New York: Association for Computing Machinery) pp 17–28
[20] Lu Y, Yamazaki I, Ino F, Matsushita Y, Tomov S and Dongarra J 2020 Reducing the amount of out-of-core data access for gpu-accelerated randomized svd *Concurrency and Computation: Practice and Experience* **32(19)** e5754
[21] Tomás A E, Rodríguez-Sánchez R, Catalán S, Carratalá-Sáez R, Quintana-Ortí E S 2019 Dynamic look-ahead in the reduction to band form for the singular value decomposition *Parallel Computing* **81** 22–31
[22] Wu R 2019 Dynamic scheduling strategy for block parallel cholesky factorization based on activity on edge network *IEEE Access* **7** 66317–66324
[23] Wu R 2018 A heterogeneous parallel cholesky block factorization algorithm *IEEE Access* **6** 14071–14077
[24] Tapia-Romero M, Meneses-Viveros A, Hernández-Rubio E 2020 Parallel qr factorization using givens rotations in mpi-cuda for multi-gpu *(IJACSA) International Journal of Advanced Computer Science and Applications* **11(5)** 636–645
[25] Islam M S and Wang Q 2020 Hierarchical jacobi iteration for structured matrices on gpus using shared memory *arXiv* **2006.16465** 1
[26] Yang X, Wang X, Sheng J, Li Y and Luo P 2018 Parallelization and performance optimization of the jacobi stencil algorithm *International Conference on Sensing,Diagnostics, Prognostics, and Control (SDPC)* (Xi'an: IEEE)
[27] Aslam M, Riaz O, Mumtaz S and Asif A D 2020 Performance comparison of gpu-based jacobi solvers using cuda provided synchronization methods *IEEE Access* **8** 31792–31812
[28] Naik T U and Guinde N 2017 Implementing the gauss seidel algorithm for solving eigenvalues of symmetric matrices with cuda *International Conference on Computing Methodologies and Communication (ICCMC)* (Erode: IEEE) pp 922–925
[29] Wu Z, Xue Y, You X and Zhang C 2017 Hardware efficient detection for massive mimo uplink with parallel gauss-seidel method *22nd International Conference on Digital Signal Processing (DSP)* (London: IEEE)
[30] Huang G H, Xu Y Z, Yi X W, Xia M, Jiao Y Y and Zhang S 2020 Highly efficient iterative methods for solving linear equations of three-dimensional sphere discontinuous deformation analysis *International Journal for Numerical Analytical Methods Geomechanics* **44(9)** 1301–1314