A Parallel Approach to Blind Source Separation in Brain Computer Interface

Titus Nanda Kumara, Geesara Prathap, Asiri Nanayakkara and Roshan Ragel
Department of Computer Engineering
Faculty of Engineering
University of Peradeniya
Peradeniya 20400
Sri Lanka
Email: ggeesara@gmail.com

***Abstract*—The original interest of Independent Component Analysis (ICA) was to deal with problems that are related to the cocktail-party problem which is a kind of blind source separation problem. Subsequently there were lots of interest in ICA and it has become clear that ICA has a lot of other interesting applications such as electrical recordings of brain activity as a representation of electroencephalogram (EEG). This situation is almost identical to the cocktail-party problem.**

When considering different implementations of ICA, different implementation gives different performance and accuracy for the user. The main goal of this project was to improve the performance by throughput while maintaining the original accuracy. Because in Brain Computer Interface, there would be some data samples that needs more computation power. To improve the performance, our main approach was to run the algorithm in parallel.

For achieving this goal, we used data-level parallelism, thread-based parallelism and parallel processing with MPI (Message Passing Interface), hybrid (threads+MPI) approach. With theses approaches, we could achieve 176x performance improvement for 118 sources and 15000 sample dataset compared to the non-parallel version.

I. Introduction

A. Independent Component Analysis

The basic problem of ICA can be described as follows. If there are n number of samples for every m number of independent sources, all the source signal would be a matrix s. These sources will be observed as matrix x after mixed by mixing matrix A. What we observe can be shown as,

$$x = As$$

Since we do not know both A and s an estimation should be done to retrieve the sources. If we can estimate the inverse of mixing matrix A as W we can separate the sources by the following equations. [2]

$$x = As$$

$$A^{-1}x = A^{-1}As$$

$$s = Wx$$

since

$$W = A^{-1}$$

W is called as a un-mixing matrix. ICA describes the procedure of estimating this W.

There is a verity of ICA algorithms out there differentiated by the procedure. A-ICA, FastICA and EASI ICA are [1] some of them which uses different approaches to the same problem. They use different procedures for obtaining linearly independent sources. The Algebraic ICA (A-ICA) algorithm is based on the algebraic dot product and the dot product is between two n-d data vectors considered as a distance measure. FastICA is a fixed point iterative algorithm. It uses a nonlinear function such as $f(y) = tanh(y)$ or $f(y) = cube(y)$, which use to applied to the separation matrix. The (EASI-ICA) Equivariant Adaptive Separation via Independence uses the equivariant batch estimators. [1]

Each algorithms' effectiveness and suitability depends on the following criteria. [1]

1) Convergence speed
2) Visible separation
3) Mixing matrix - separated matrix product error
4) Separated matrix - mixing matrix difference error

TABLE I
Comparison of convergence speed and visible and numeric accuracy for each ICA

Algorithm	Convergence speed	Visibly Accurate	Numeric accuracy
A-ICA	Fast	No	Low
Fast-ICA	Medium	Yes	High
EASI-ICA	Medium	Yes	High

Comparison of different ICA algorithm are listed in Table I. Numeric accuracy was decided considering both product and difference between separated matrix and mixing matrix as mentioned above.

Considering all the factors we had to choose one of the algorithms between EASI-ICA and FastICA we chose FastICA it has more resources available against EASI-ICA.

II. FAST-ICA

A. Algorithm

This algorithm is used to separate mixed sources using maximizing non-gaussianity. Hence our problem is

$$x = As$$

where x is observation or data matrix, s is the original components and A is the transformation matrix that defines the linear mixing of the constituent signals. Here A and s are unknown. FastICA assumes that x which is observation data has been centered and whitened. Which mean that there should be some pre-processing has to be done in order to make this happens. There are two main pre-processing in FastICA, centering and whitening/sphering. Reasons for doing pre-processing are [3]

- Make algorithm as simple as possible
- If there are some dependent components in input data-set those data has been removed and get independent components only
- Decrease number of calculations
- Because of finding out mean and covariance matrix for input data set, get clear idea about relations between sample dataset centering data

B. Preprocessing

If input matrix is X which is composed of n number of samples and K number of sources were mixed. In this stage mean of input matrix is subtracted from each components of input matrix. Next step is to remove any correlations in that dataset. One popular method for doing that is to use the Eigen Value Decomposition (EVD) of the covariance matrix of input matrix. In this implementation SVD (Single Value Decomposition) is used for EVD. Result of SVD gives eigen vector (u) and eigen values(d) as diagonal matrix. We need to find matrix (k) such that transpose of (u/v). Then dot product of k and x will resulted another matrix X1. Then X1 should be multiplied by square-root of number of samples.

After doing the pre-processing, now input data(X1) has been whitened X and now FastICA algorithm can be applied to separate each source from mixing data set. This process is called single component extraction in case only separates one single component and multi-component extraction is used to separate multiple components.

C. FastICA main loop

After doing all those pre-processing this iterative algorithm can be applied to find the direction for the weight vector W minimizing the gaussianity of the projection $W^T X$ for the data T. The function $g(u)$ is the derivate of a non-quadratic nonlinearity function $f(u)$. Hyvrinen and Oja [2] states good equation for f (shown with their derivatives g and second derivatives $g^{'}$) are

$$f(u) = logcosh(u); g(u) = tanh(u); g'(u) = 1 - tanh^2(u)$$

According to [2] following pseudo code will generate the solution for FastICA

for p in 1 to C:
$\mathbf{w_p} \leftarrow$Random vector of length N
while $\mathbf{w_p}$ changes
$\mathbf{w_p} \leftarrow \frac{1}{M}\mathbf{X}g(\mathbf{w_p}^T\mathbf{X})^T - \frac{1}{M}g'(\mathbf{w_p}^T\mathbf{X})\mathbf{1w_p}$
$\mathbf{w_p} \leftarrow \mathbf{w_p} - \sum_{j=1}^{p-1} \mathbf{w_j}\mathbf{w_p}^T\mathbf{w_j}$
$\mathbf{w_p} \leftarrow \frac{\mathbf{w_p}}{\|\mathbf{w_p}\|}$
Output:

$$\mathbf{W} = \begin{Bmatrix} w_p \\ \vdots \\ w_C \end{Bmatrix}$$

$$\mathbf{S} = \mathbf{WX}$$

III. IMPLEMENTATION OF C++ SOLUTION

For the implementation of C++ solution, we used python FastICA implementation which is already available in the scikit-learn package [4] as a reference code. For matrix manipulations, we chose Eigen C++ library since it five is a template library and it is lightweight. In the FastICA implementation, it has five stages 1) Reading data from file. 2) Pre-processing. 3) FastICA iterations until the result is converged. 4) Calculate original sources. 5) Data writing to a file

With different samples we measured the execution time for these stages. Figure 1 shows the execution time for each stage as a percentage value.

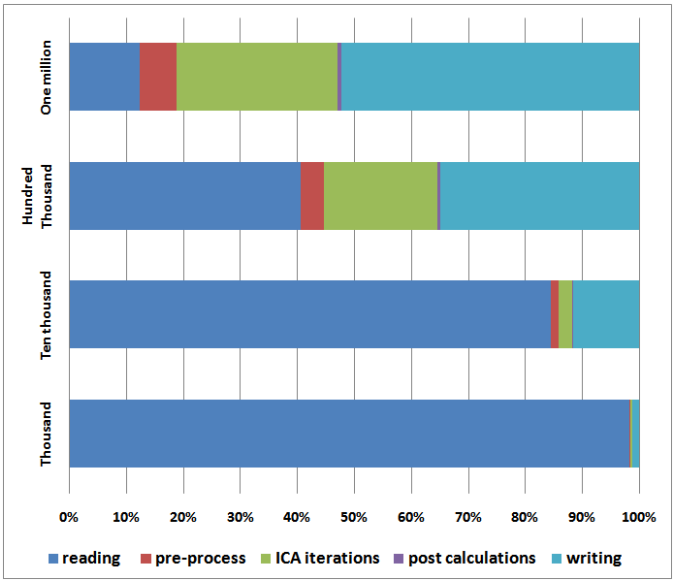

Fig. 1. Percentage time consumption for each stages of FastICA algorithm for three sources with different sample size

as shown is Figure 1 it is clear that when the sample size is increasing the computational time consumes higher proportion. Since we are going to improve the performance of FastICA algorithm, we can do it by improving the performance of FastICA iterations.

IV. Parallelizing FastICA

In a real word scenario when testing data which is taken from electrodes which are on top of the human scalp may have to take about five seconds to five hours depending on the situation. In that case, analysis those data is difficult with the same algorithm or the machine. Sometimes processing power is not enough to give enough speed as well as memory would be not sufficient to handle large data set. Therefore, we need to address those problems in the proper way. But whatever approaches are used, the final outcome should be the reduce execution time for separate sources from mixed sources. When it comes to real-time processing, time is the main constraint. Because everything is decided depends on the result of the algorithm.

According to the requirement, the way it should be processed should be good enough to get better performance. In our approach, four models were implemented in order to get better performance in difference situations. Those approaches are MPI implementation, multithread implementation, a combination of both MPI and Threads and CUDA implementation.

MPI implementation: This is mainly targeted for the testing dataset which is large enough, hence cannot handle in a single node (a computer) as well as which required more processors to process. Because main memory in a single machine may not enough to store whole testing data. On the other hand when testing data set is comparatively small and not required more processing power, multithreaded approached can be applied. Though MPI is suitable for processing when data size is big. It takes lots of time for intercommunication among processes. To reduce network communication time among MPI processes as well as to supply good processing power and memory for processing, MPI and Threading combined approach can be used. In this approach, one MPI process is allocated for one node (a computer). Inside each MPI process, there would be some user desired number of threads can be run. This number basically depends on the number of cores is available in the relevant machine.

The architecture of Data parallelism is shown in Figure 2 As shown in Figure 2 whole data set devided into n number of samples. Then the algorithm feed the relevant data into relevant n number of threads. The data is synchronized in each loop iteration.

V. Results

A. Verifying the parallel implementations

There are two main objectives which need to be satisfied in this implementation. One is correctness of the algorithm and accuracy of the result of the algorithm. The first approach is used to measure the correctness of the algorithm. It was, creating sample signal set which are having some considerable amount of samples such that it may varied 1000 to 1000000. Next those original sources are mixed each other according to some distribution (beta/chique). The reason for using these kind of distribution other than random distribution is when the data is generated from random distribution and size of

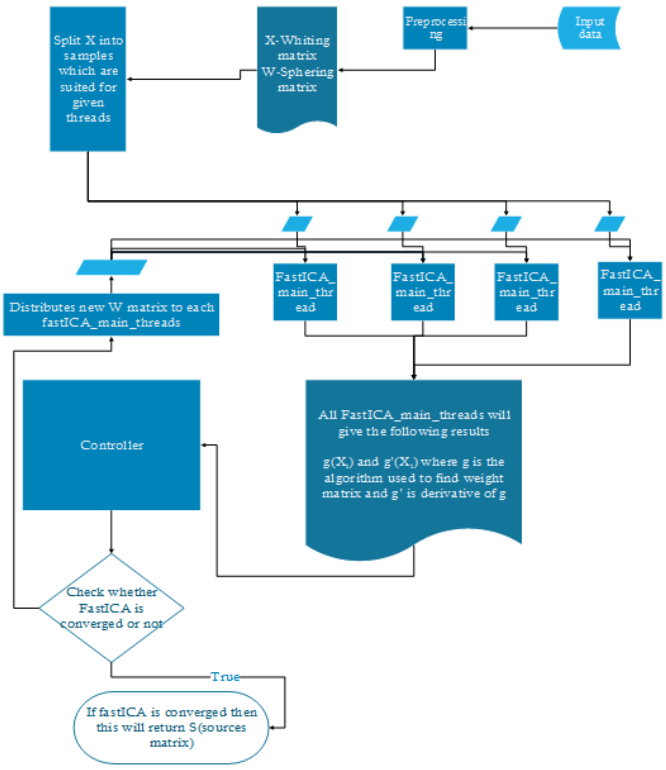

Fig. 2. Data level parallelism on fastica

data set is increasing those values eventually go to normal distribution according to central limit theorem. To avoid that problem above mentioned different distributions were used. Next step was to apply FastICA algorithm to extract original sources from mixing signals. Now there is two type of signal sets which are actual signal set and extracted signal set using FastICA algorithm. So we can compare those two signal set and check the accuracy and find reasonable threshold.

B. Performance improvement

We used different machines to test our solution.

TABLE II
Specifications of test machines

Machine	Clock Speed	Cache size	Memory
(Single machine) Intel i5-3470	3.20GHz	6MB	4GB
(High-performance computer) Intel Xeon E5-2670	2.60GHz	20MB	256GB

For testing purpose, a cluster with 2 single machines were used.

The algorithm was developed in different layers and different stages while improving performance. Figure 3 shows the different implementations and the tested machines. S-Single machine, H-High performance computer, C-Cluster with two machines.

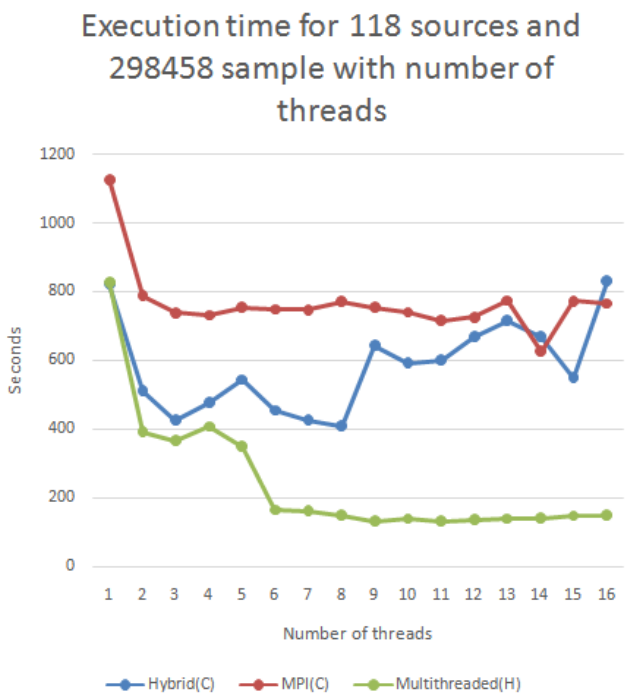

Fig. 4. Execution time for FastICA when sample size is 298458 with 118 sources

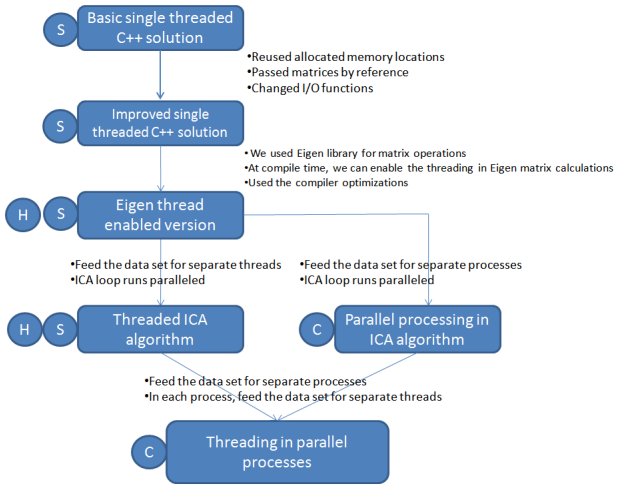

Fig. 3. Different stages of FastICA improvement and tested machines

C. Execution time for different implementations

After verifying the accuracy of the algorithm, we applied FastICA for real EEG signals which are coming from brain. In this implementations, we used the data level parallelized version which uses Eigen threading also. Sample test data is taken from BCI competition 3 [5]. Test data which consist of 118 channels and 298458 samples.

As the Figure 4 shows, execution time is decreasing with the number of data portions we feed into the algorithm.

D. Final result

We tested with actual 118 source, 15000 sampled data set [5] in each implementation and recorded the best FastICA execution time for each version runs on each machine. The version with no data level parallism and thread level paralalism took more than 4060 Seconds to finish the execution. But when enabling Eigen threading,even the slowest version (runs on single machine) takes about 38 seconds to finish the execution. This is about 106x speed up. With the best version (Thread and data parallelism on High performance computer) gains about 176x speed up.

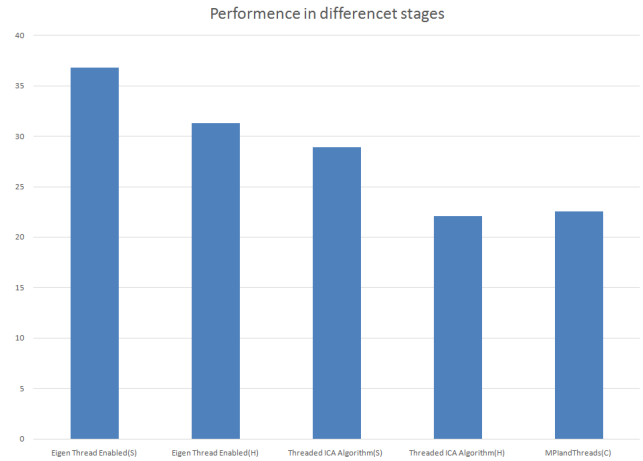

Fig. 5. Improvement and tested machines (Without the serial version)

VI. Conclusion

With the experiments and results we performed for different implementations of FastICA we observe that the parallelism is a good way to improve the performance of FastICA algorithm.

There is two type of parallelism can be achieved in this implementations. Data level parallelism and thread level parallelism. Data level parallelism can be achieved sample data set is divided into separate parts according to the number of threads provided. Next method is thread level parallelism. We achieved this by using the Eigen library.

With both thread level parallelism and data level parallelism, we could achieved more than 100 time speedup compared to the serial version. This proves that the FastICA is parallel-able and we can gain high speedup if we use parallelism.

References

[1] Odom Crispin, *Independent Component Analysis Algorithm Fpga Design To Perform Real-Time Blind Source Separation.* The Florida State University, 2012.
[2] Aapo Hyvrinen and Erkki Oja *Independent Component Analysis: Algorithms and Applications.* Helsinki University of Technology P.O. Box 5400, FIN-02015 HUT, Finland, 2000.
[3] Li and J. Zhang *Sphering and its properties* The Indian Journal of Statistics,1998.
[4] Scikit-learn *http://scikit-learn.org/stable/*
[5] BCI competition 3 *http://bbci.de/competition/iii/descIVa.html*