Performance comparison of parallel fastICA algorithm in the PLGrid structures

Anna Gajos-Balinska[1*], *Grzegorz M.* Wojcik[1], *Przemyslaw* Stpiczynski[2]

[1]Department of Neuroinformatics, Institute of Computer Science, Maria Curie-Sklodowska University, ul. Akademicka 9, 20-033 Lublin, Poland
[2]Institute of Mathematics, Maria Curie-Sklodowska University, Plac Marii Curie-Sklodowskiej 1, 20-031 Lublin, Poland

Abstract. During processing the EEG signal, the methods of cleaning it from artifacts play an important role. One of the most commonly used methods is ICA (independent component analysis) [1-3]. However, algorithms of this type are computationally expensive. Known implementations of ICA type algorithms rarely include the possibility of parallel computing and do not use the capabilities provided by the architecture itself. This paper presents a parallel implementation of the fastICA algorithm using the available libraries and extensions of the Intel processors (such as BLAS, MKL, Cilk Plus) and compares the execution time for two selected architectures in the PLGrid structure (Zeus and Prometheus).

1 Introduction

Electroencephalography as a non-invasive diagnostic method has been used for many years both in medicine and in experimental psychology. For this purpose, various methods of signal analysis and processing are applicable [4-5]. The recorded signal of brain activity is easily disturbed by external events, therefore an important element of working with the EEG signal is to clean it from artifacts [6].

Another important element of signal analysis is the extraction of features for specific stimuli in ERP experiments (event-related potentials). A very often used method that accomplishes both tasks is the so-called blind source separation, while independent component analysis (ICA) is one of these methods [7].

While working with various types of EEG machines, it could be noted that the EEG system manufacturer does not always provide the right tools for effcient and desired processing. It is common that one should use external software to take advantage of the aforementioned method. This prompted the authors of the research to write an application with its own implementation of ICA, integrated with the software used in the Department of Neuroinformatics at Maria Curie-Sklodowska University in Lublin (NetStation) [8–11].

However, the algorithm itself is computationally expensive (as an iterative algorithm that uses the Newton approximation method) [12]. It can take many hours to process one data

* Corresponding author: agajos@hektor.umcs.lublin.pl

recording, which is bothersome for both the researcher and the patient who is waiting for results. Therefore, the next step was to develop a method of time optimization of the selected ICA algorithm by using the capabilities of modern parallel architectures. The fastICA algorithm [13] was adopted as a model in studies. In this paper, a comparison of the time efficiency of the algorithm for two selected machines made available by PLGrid (Zeus and Prometheus) will be presented. The main purpose of this research is to develop a faster method of preprocessing EEG data.

2 Independent Component Analysis and fastICA implementation

2.1 The ICA method

The recorded EEG signal is, in fact, a mixture of many signals coming from different sources. If on the basis of these mixtures one can separate the source signals from each other, this is called blind source separation (BSS) [12, 14].

We can illustrate this problem using the equation:

$$\mathbf{S} = \mathbf{WX}, \tag{1}$$

where $\mathbf{S} \in \mathbf{R}^{\mathbf{CxM}}$ is the matrix of C components for M samples, $\mathbf{W} \in \mathbf{R}^{\mathbf{CxN}}$ is the transition matrix with the weight vectors between each signal and electrode and $\mathbf{X} \in \mathbf{R}^{\mathbf{NxM}}$ is the data from N electrodes.

The problem is to find a separating matrix **W** that satisfies this equation. However, it should be noted that no more sources can be found than the number of signal mixtures (N) and that the found sources will have a similar shape, but it is not possible to find the original amplitude.

The ICA discussed, in this case, is one of the approaches to blind source separation. Unlike other BSS methods, ICA is based on higher-order statistical methods. It should be noted that the use of statistical methods results in one more disadvantage: if the primary signals have a distribution close to normal, the result is ambiguous. ICA is based on the assumption created on the central limit theorem, that the signal sources are independent components, and the mix of these signals has a normal distribution. Therefore the task of the algorithm is to separate the independent components [12].

For this purpose, the data is preprocessed, i.e. centering (the mean of each signal is zero) and whitening (the variance of each signal is equal to 1, this is done by decomposition into eigenvectors and eigenvalues of the matrix). As a result, uncorrelated signals are received, but they are not yet independent of each other.

Finally, one needs to find such weights (**W** matrix) so that the received signals have the least normal distribution. Different measures of normality (negentropy and kurtosis) are used, and weights are modified using Newton approximations using non-quadratic functions [14]. In our case, the hyperbolic tangent function was used.

As previously mentioned in the work, the fastICA algorithm was used, as a very common implementation of ICA algorithm. It is relatively stable, it is characterized by quick convergence, and the source code is available. In addition, its design allows using parallel computations. It differs from other algorithms by way of weight modification [12].

2.2 Implementation and data representation

The implementation of the fastICA algorithm presented in this paper is based on the version that can be found in the open library it++ and in MATLAB. In contrast, the implementation

discussed in this work does not use the reduction of the matrix dimensions, which increases the accuracy of calculations. During subsequent modifications of weights, the tanh function (hyperbolic tangent) from the standard mathematical library is used. To prevent the so-called cache misses, the data is properly arranged and the Intel mm malloc compiler function is used.

In the parts of the code where the entire signal is used, parallel blocks are applied. With the most time-consuming parts of the algorithm, i.e. matrix multiplication or eigenvectors calculation, functions from the BLAS and MKL libraries are used.

Thanks to the Intel Cilk Plus extensions for C and C++, it is possible to use array notation and built-in reduction functions (such us searching maximum in the array) that not only make the code more transparent but most of all force the effective vectorization [15].

3 Results

All tests were performed on two architectures:

1. Zeus: Intel Xeon X5650 2.67GHz – 12 cores (+12 virtual)
2. Prometheus: Intel Xeon E5-2680 2.5GHz – 24 cores (+ 24 virtual)

Prometheus and Zeus are superfast computers providing computing servers as part of the ACK Cyfronet AGH platform. In the latest release of TOP500 list, Prometheus was classified on 77th position. It has over 53 000 computing cores. Zeus was listed on the TOP500 list twelfth in a row and for many years was the fastest supercomputer in Poland. It provides over 25 000 computing cores.

A test comparing the speed of fastICA implementation for 1 second, 10 seconds, 100 seconds and 1000 seconds of the signal recording was performed. For both architectures, the time performance was compared depending on the number of threads for which the program was run.

In Table 1 and Table 2 there is the program execution time for all data in seconds. The Figure 1 and Figure 2 show the speed-up performance of multithreaded launching of application for both architectures compared to single-threading launching. The application of the aforementioned methods (parallel blocks, vectorization, functions from BLAS and MKL libraries) has increased the speed of calculation compared to the single-threaded version. It can be seen in Figure 3 that parallel implementation is scalable and the increase in data size generates more profits. However, the physical number of threads matters. Using more threads than their physical amount (24 threads for Zeus and 48 for Prometheus) does not generate more and more profit. It can also be seen that the efficiency in Zeus decreases after exceeding the physical number of threads. At Prometheus, it is kept at the same level. This may mean that newer architectures cope better with hyperthreading.

Table 1. Zeus.

Number of threads	1 second	10 seconds	100 seconds	1000 seconds
1	7.853036	24.3698	249.89457	2654.8714
2	5.59254	15.7137	152.722401	1658.469306
4	2.653978	11.044857	97.381287	714.447443
6	2.311765	8.702663	67.243015	616.629619
8	2.379572	5.509977	49.668591	476.512014
10	2.931615	5.529254	44.437239	460.055876
12	2.132589	5.374643	46.453258	419.557943
14	3.816939	9.196031	53.868611	449.6515
16	4.867699	8.662239	56.572302	440.946023
18	4.902744	8.41807	50.672497	684.304196
20	5.509669	9.079854	76.22381	495.427938
22	5.153895	9.94856	78.739455	634.250165
24	5.3544	10.513036	64.507768	669.666002

Table 2. Prometheus.

Number of threads	1 second	10 seconds	100 seconds	1000 seconds
1	2.712171	10.101657	122.676974	1229.070627
2	2.382829	7.668297	96.769035	947.622293
4	1.941356	5.153685	61.690553	639.179694
6	1.930588	4.479713	44.180151	437.537964
8	1.876067	3.467063	38.85796	372.240371
10	1.841483	3.101005	34.147887	386.33065
12	1.388474	2.916747	30.698477	268.486074
14	1.416901	2.871293	26.730792	274.935783
16	1.304905	2.20292	27.630703	242.159316
18	1.525153	2.215691	27.36333	249.796984
20	1.459399	2.804208	24.865453	193.747517
22	1.629841	2.787488	25.010329	229.511152
24	1.853309	3.124078	24.873183	170.942183
26	2.167185	3.659332	27.361812	204.138773
28	2.935814	3.502416	26.860298	239.987754
30	2.556604	3.5886	28.249686	226.862231
32	3.357873	4.52818	28.521366	217.752913
34	3.56577	5.151289	30.850415	240.55447
36	4.140595	4.335072	34.031625	212.100187
38	3.13482	4.85156	28.672792	237.856584
40	4.259223	5.806984	29.720013	292.608695
42	4.067353	5.023907	36.669899	235.577985
44	4.78252	5.347281	36.054015	302.960874
46	4.877022	5.014137	33.573492	308.539336
48	4.414425	6.715821	41.665372	272.523367

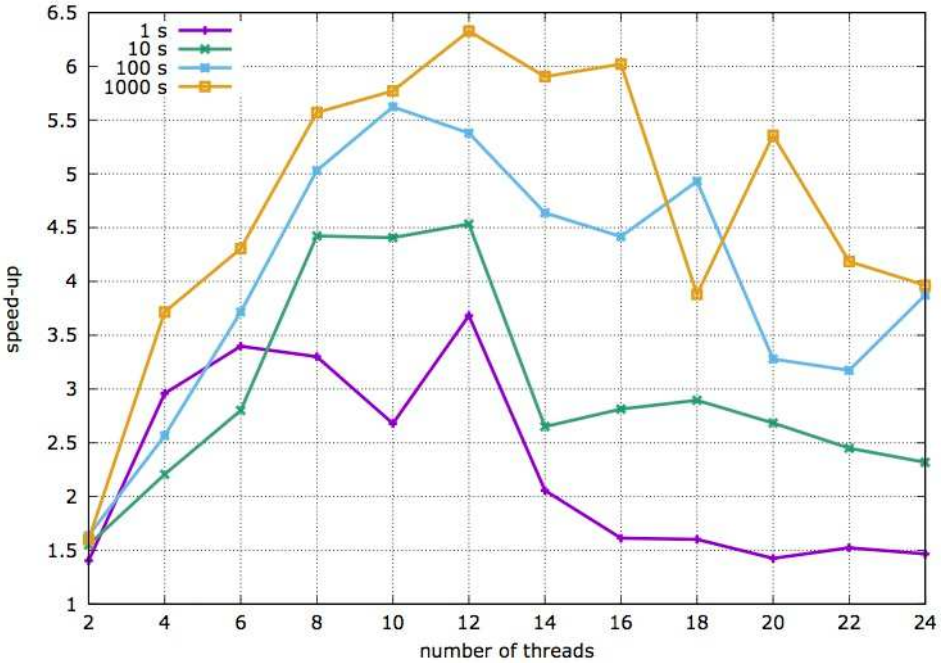

Fig. 1. Speed-up of multithreaded launching – Zeus

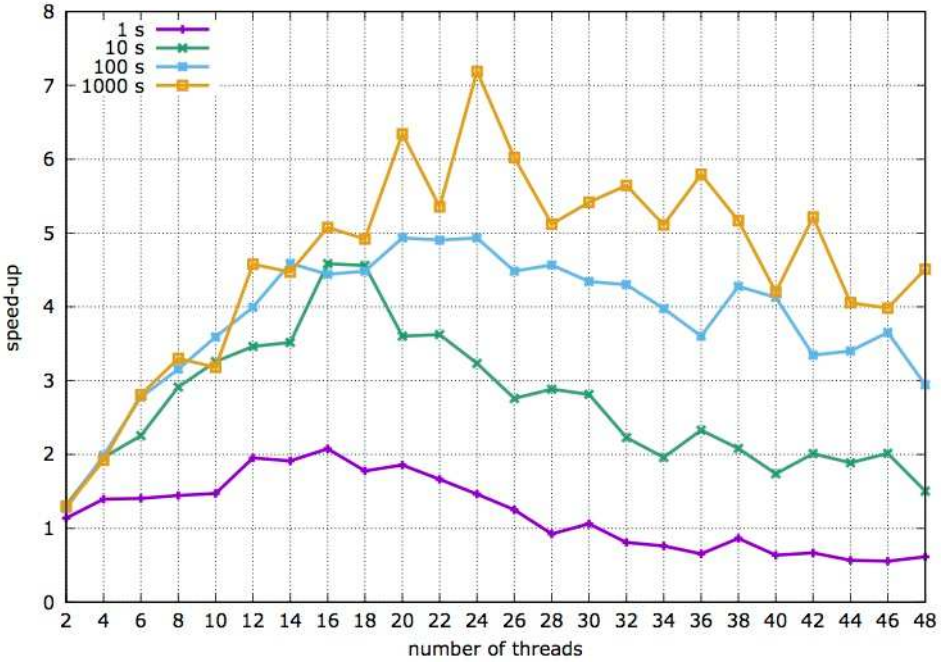

Fig. 2. Speed-up of multithreaded launching – Prometheus

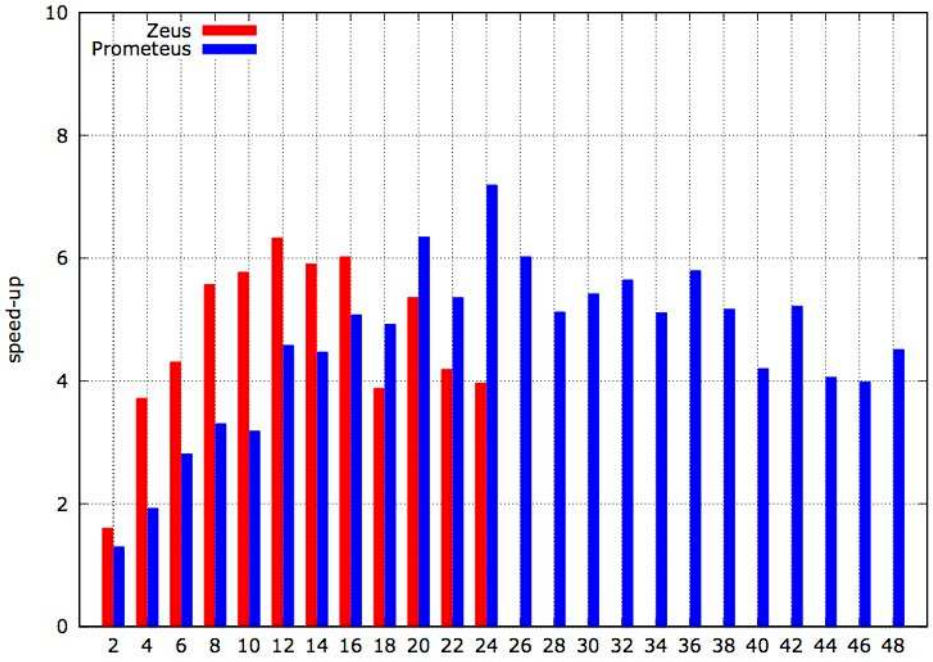

Fig. 3. Speed-up comparison for Zeus and Prometheus architecture

4 Conclusions

The obtained results showed that the execution time scales with the increase of the problem. It can also be seen that on Prometheus (a machine with a newer architecture) the optimal number of threads increases with the size of the problem, while in Zeus it is constant (12 threads).

The studies have shown that architecture and the physical number of cores have a fundamental impact on the speed of calculation in the parallel version of fastICA. The newer generation machine performs better with hyperthreading and it is cost-effective to use the capabilities of the latest generation processors and parallelism.

5 Summary

This paper presents a parallel version of the fastICA algorithm. It includes the capabilities of multicore Intel architecture and processor. The algorithm has been applied to real EEG data. It was launched on two machines (Zeus and Prometheus in the PLGrid structure).

In addition, the number of physical cores is important, because efficiency decreases (in the case of Zeus) or does not grow (in the case of Prometheus) if we run the program for the number of threads exceeding the number of physical cores.

The future plan is to integrate our solution with tools for EEG signal processing such us NetStation [8-11]. It is also worth considering the adjustment of the existing solution to the capabilities of a particular architecture to achieve even better results.

We have an experience in parallel computing when we modelled some cognitive processes occurring in the simulated cortex of mammalian brains [16–19]. From our point of view, it is interesting if it is possible to find such methodology for algorithm optimization so that the given architecture could achieve maximum performance.

This research was supported in part by PLGrid Infrastructure. Computations have been performed at ACC Cyfronet AGH.

References

1. G.D. Brown, S. Yamada, T.J Sejnowski. Trends in neur., **24**, 54 (2001)
2. A. Delorme, T. Sejnowski, S. Makeig. Neuroimage, **34**, 1443 (2007)
3. M. Ungureanu, C. Bigan, R. Strungaru, V Lazarescu. Measur. Science Review, **4**, 18 (2004)
4. R. Tadeusiewicz. *Computers in psychology and psychology in computer science.* (2010)
5. R. Tadeusiewicz. *Using neural networks for simplified discovery of some psychological phenomena* (2010)
6. C.L. Dickter, P.D. Kie□aber. *EEG Methods for the Psychological Sciences* (2014)
7. A. Albajes-Eizagirre, L. Dubreuil Vall, I. David, A. Riera, A. Soria-Frisch, S. Dunne, G. Ru□ni. *EEG/ERP Analysis: Methods and Applications* (2014)
8. A. Gajos, G.M. Wojcik. Bio-Alg. and Med-Sys., **12**, 67 (2016).
9. A. Gajos-Balinska, G.M. Wojcik, P. Stpiczynski. CGW'15 (2015)
10. A. Gajos-Balinska, G.M. Wojcik, P. Stpiczynski. CGW'17 (2017)
11. A. Gajos-Balinska, G.M. Wojcik, and P. Stpiczynski. Lect. N. in Comp. Sci. **10777**, 495 (2018)
12. A. Hyvarinen, E. Oja. Neural networks, **13**, 411 (2000)
13. G. Sahonero-Alvarez, H. Calderon. IMCIC'17 (2017)
14. A. Hyvarinen, IEEE Transactions on Neural Networks, **10**, 626 (1999)
15. R. Rahman. *Intel Xeon Phi Coprocessor Architecture and Tools: The Guide for Application Developers*. (2013)
16. M.Wazny, G.M. Wojcik. Neurocomputing, **135C**, 139 (2014)
17. G.M. Wojcik, J.A. Garcia-Lazaro. Procedia Computer Science, **1**, 845 (2010)
18. G.M. Wojcik, W.A. Kaminski. Neurocomputing, **70**, 2593 (2007)
19. G.M. Wojcik, W.A. Kaminski, P. Matejanka. In Lect. N. in Comp. Sci. Springer **4671**, 468 (2007)