2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC) | 978-1-6654-3860-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/MCSOC51149.2021.00033

A Computation-Aware TPL Utilization Procedure for Parallelizing the FastICA Algorithm on a Multi-Core CPU

Lan-Da Van, Tao-Jung Wang, Sing-Jia Tzeng
Department of Computer Science,
National Yang Ming Chiao Tung University, Taiwan.
ldvan@cs.nctu.edu.tw; tjwang.cs08@nycu.edu.tw;
tzengsj@gmail.com

Tzyy-Ping Jung, Fellow, IEEE
Swartz Center for Computational Neuroscience,
University of California, San Diego (UCSD), USA.
tpjung@ucsd.edu

***Abstract* – Independent Component Analysis is a widely used machine learning technique to separate mixed signals into statistically independent components. This study proposes a computation-aware (CA) Task Parallel Library (TPL) utilization procedure to parallelize the Fast Independent Component Analysis (FastICA) algorithm on a multi-core CPU. The proposed CA method separates the complex from simple computations by exploring their execution times on a multi-core CPU. TPL is used for complex calculations, but not for simple ones. In comparison to the program without the TPL, the proposed CA procedure reduces the execution time of decomposing 8- and 32-channel artificially mixed signals by 34.88% and 43.01%, respectively. The proposed CA procedure reduces the execution time of decomposing 8- and 32-channel artificially mixed signals by 10.04% and 0.93%, respectively, compared to the fully parallelized program with TPL. Using CA TPL, the decomposition of 12-channel electroencephalograms (EEG) signals take 48.27% less time than without it. The proposed CA procedure reduces execution time by 15.12% compared to the fully parallelized program with TPL.**

Keywords – Computation-aware, FastICA, multi-core, parallel, TPL.

I. Introduction

Electroencephalography (EEG) has been widely used in biomedical, clinical, medical, and science fields; however, the EEG signals measured by the sensors are the mixture of various sources within the brain. Among these sources, the artifacts such as eyeblinks and heartbeats contaminate brain activities. These artifacts pose great challenges to the interpretation of brain activities.

The independent component analysis (ICA) algorithm [1-17], which belongs to the field of machine learning [18, 19], is commonly used to separate mixed signals into independent components. Through the ICA processing, the artifacts can be separated from the EEG signals such that the pure brain activities can be retrieved. In other words, the ICA algorithm can estimate the original source signals by processing the mixed signals from the scalp surface. Many ICA algorithms and implementation are proposed in recent years [1-17]. In [5], a comparative study of ICA algorithms for brain-computer interface (BCI) is explored. According to [5], the FastICA algorithm [2, 4] shows good separation quality as well as low complexity among the ICA algorithms. On the other hand, although the FastICA algorithm has relatively lower complexity than other ICA algorithms, the computation of the FastICA algorithm still not very efficient. Several speedup solutions have been developed to reduce the calculation time of FastICA, including application-specific hardware approaches in ASIC/FPGA [6-9], general-purpose processor and parallel processing software approaches in CPU/GPU/GPGPU [10-16], and collaboration of application-specific hardware and general-purpose processor approach [17]. Even hardware approaches can speed up the FastICA processing in real-time but request more hardware resources and show less flexibility. For the general-purpose processor and parallel processing software approaches, there are many possible choices for the implementation. Many methods, libraries, and frameworks were proposed for parallel programming such as Open Multi-Processing (OpenMP) [20], Message Passing Interface (MPI) [21], Compute Unified Device Architecture (CUDA) [22], Open Computing Language (OpenCL) [23-24], and Task Parallel Library (TPL) [25-26]. To our best knowledge, we are the first to apply TPL to enhancing the FastICA algorithm [13]. This is because we observed that not all the codes are suitable for parallelization via TPL. If the codes only contain a small amount of computation, the overhead of parallelization will increase instead of reducing the execution time. Thus, this study aims to provide a computation-aware (CA) TPL mechanism to optimize the FastICA processing time by exploring different levels of computation loads. The article is organized as follows. Section II briefly reviews the FastICA algorithm. Section III proposes the CA TPL utilization procedure for the FastICA algorithm. Section IV presents the simulation and comparison results. Section V provides the concluding remarks.

II. Review of FastICA Algorithm

This section briefly reviews the FastICA algorithm [2, 4]. The FastICA algorithm has been widely used for the blind source separation (BSS) problem as defined in (1) [6, 7]:

$$\mathbf{X} = \mathbf{AS} \tag{1}$$

where $\mathbf{A}$ is an $n \times n$ mixing matrix, $\mathbf{X}$ is a matrix with n observed mixed signal vectors and $\mathbf{S}$ is a matrix with n blind source signal vectors that are statistically independent and no more than one signal is Gaussian distributed. $\mathbf{X}$ and $\mathbf{S}$ are expressed as follows:

$$\mathbf{X} = [\mathbf{x}_1\ \mathbf{x}_2 \dots \mathbf{x}_n]^{\mathrm{T}} \tag{2}$$

$$\mathbf{S} = [\mathbf{s}_1\ \mathbf{s}_2 \dots \mathbf{s}_n]^{\mathrm{T}} \tag{3}$$

where $\mathbf{x}_i$ and $\mathbf{s}_i$ are denoted as follows:

$$\mathbf{x}_i = [x_i(1)\ x_i(2) \dots x_i(m)]^T \tag{4}$$

978-1-6654-3860-5/21/$31.00 ©2021 IEEE
DOI 10.1109/MCSoC51149.2021.00033

$$\mathbf{s}_i = [s_i(1)\ s_i(2) \ldots s_i(m)]^T \quad (5)$$

where $x_i(j)$ is a mixed signal at time j, $s_i(j)$ is a source signal at time j, and m is the sample number. The goal of ICA is to recover the source signal **S**. In order to achieve the purpose, the ICA algorithm estimates the matrix **A** by observing matrix **X** and computes the weight matrix $\mathbf{W}^T$ that is equal to the inverse of matrix **A**. The FastICA algorithm contains the preprocessing and fixed-point algorithm.

A. Centering and Whitening

The preprocessing of FastICA algorithm includes two steps: centering and whitening. Centering refers to subtracting the mean value from the mixed signal such that the zero-mean signal can be obtained. The centering process can be expressed as follows:

$$\bar{x}_i(j) = x_i(j) - E\{x_i\} \text{ for } i = 1, 2, 3, \cdots, n\ , \quad (6)$$

where $E\{x_i\}$ denotes the expected value of the random variable $x_i(j)$. After the centering process, the centered matrix can be obtained as

$$\overline{\boldsymbol{X}} = \begin{bmatrix} \bar{\mathbf{x}}_1^T \\ \bar{\mathbf{x}}_2^T \\ \vdots \\ \bar{\mathbf{x}}_n^T \end{bmatrix} = \begin{bmatrix} \bar{x}_1(1) & \bar{x}_1(2) & \cdots & \bar{x}_1(m) \\ \bar{x}_2(1) & \bar{x}_2(2) & \cdots & \bar{x}_2(m) \\ \vdots & \vdots & \cdots & \vdots \\ \bar{x}_n(1) & \bar{x}_n(2) & \cdots & \bar{x}_n(m) \end{bmatrix} \quad (7)$$

Next, the whitening process is required. The whitening step transforms the vector **x** into uncorrelated with unit variance. The eigenvalue decomposition (**EVD**) can be used to decompose the covariance matrix of **x** and the corresponding operation is expressed as follows:

$$\mathbf{C_x} = E\{\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T\} = \mathbf{EDE}^T \quad (8)$$

where $\tilde{\mathbf{x}}$, **E** and **D** denotes the random column vector in $\overline{\mathbf{X}}$, the eigenvector matrix, and the eigenvalue matrix, respectively. **D** can be expressed as

$$\mathbf{D} = diag(d_1, d_2, \ldots, d_n) \quad (9)$$

where $d_1, d_2, \ldots, d_n$ denote the eigenvalues of $\mathbf{C_x}$. Finally, the whitening process of $\overline{\mathbf{X}}$ is expressed below

$$\mathbf{Z} = \mathbf{D^{-1/2}E}^T\overline{\mathbf{X}} = \mathbf{P}\overline{\mathbf{X}} \quad (10)$$

where **P** equals $\mathbf{D^{-1/2}E}^T$ and denotes the whitening matrix of $\overline{\mathbf{X}}$. Through the above whitening process, we can obtain the whitened matrix **Z**. The covariance matrix of $\tilde{\mathbf{z}}$ is identity matrix:

$$\boldsymbol{E}\{\tilde{\mathbf{z}}\tilde{\mathbf{z}}^T\} = \mathbf{I} \quad (11)$$

B. Fixed-Point Algorithm

After the preprocessing, the fixed-point algorithm is used to train the weight matrix which maximizes the non-Gaussianity. Non-Gaussianity is here measured by the approximation of negentropy *J*(*y*) [2, 4, 6, 7]

$$J(y) \propto [E\{G(y)\} - E\{G(v)\}]^2 \quad (12)$$

where G denoting a non-quadratic function is defined as follows:

$$G(u) = \frac{1}{a}\, log\ cosh(au) \quad (13)$$

where a denotes a constant parameter. Using the above equations, the FastICA algorithm can find the maximum of the non-Gaussianity by measuring the negentropy. For the fixed-point algorithm, either deflation scheme or symmetric orthogonolization can be used to estimate multiple independent components [2, 4]. The deflation scheme estimates the independent component one by one. The symmetric orthogonolization estimates the independent components in parallel. In this work, we adopt the fixed-point algorithm with symmetric orthogonolization. The procedures of the fixed-point algorithm with symmetric orthogonolization are addressed as follows:

Step 1: Choose initial (e.g. random) vector $\mathbf{w}_i$ with unit norm for *i=1, 2, 3, ..., n*.
Step 2: Calculate $\mathbf{w}_i^+ = E\{\tilde{\mathbf{z}}[g(\mathbf{w}_i^T\tilde{\mathbf{z}})]\} - E\{g'(\mathbf{w}_i^T\tilde{\mathbf{z}})\}\mathbf{w}_i$ for *i=1, 2, 3, ..., n*.
Step 3: Calculate $\mathbf{W} = [\mathbf{W}^+(\mathbf{W}^+)^T]^{-1/2}\mathbf{W}^+$.
Step 4: If not converge, go back to Step 2

where *g* is the derivative of the non-quadratic function *G* and $\mathbf{W}^+ = [\mathbf{w}_1^+\ \mathbf{w}_2^+\ \ldots \mathbf{w}_n^+]$. When the convergence is satisfied, we can obtain the weight matrix **W** to estimate the source signals as $\mathbf{S} = \mathbf{W}^T\mathbf{Z}$. However, the long execution time due to high computation for the FastICA algorithm still exists. Therefore, the parallelization is necessary for the FastICA algorithm in PC to improve the speed. The details will be discussed in the next section.

III. Computation-Aware TPL Utilization Procedure for FastICA Algorithm

A. Tested Matrix Operations

Because the FastICA algorithm mainly utilizes matrix operations, we focus on how to improve the performance of the matrix operations. To demonstrate the capability and limitation of TPL, we test the execution times of the matrix operations with different matrix dimensions. The matrix operation is coded in a for loop way with TPL and without TPL in C# [27]. We divide the matrix operation into two types: simple computation and complex computation. The matrix operation is considered as a simple computation when the execution time of the matrix operation without TPL is shorter than that with TPL. Otherwise, the matrix operation is considered as a complex computation. The tested matrix operations included matrix multiplication, matrix addition, matrix divided by a constant, hyperbolic tangent. We use the stopwatch class [28] to measure the execution time on a PC featuring Intel Core 2 Quad Q8200 (4 physical cores). Tables 1~5 summarize the execution times. $\mathbf{A}_{i\times j}$ and $\mathbf{B}_{i\times j}$ denote two $i \times j$ matrices and c denotes a constant. Table 1 shows the execution times of the multiplication of two square matrices. As can be seen, We set the threshold of matrix multiplication at $\mathbf{A_{16\times16}} \times \mathbf{B_{16\times16}}$. In other words, if two square matrices whose dimensions are equal

to or higher than 16, their multiplication is regarded as a complex computation. Otherwise, the matrix multiplication is regarded as a simple computation. Table 2 shows the execution times of the multiplication of a square matrix and a rectangular matrix. When the matrices are equal to or larger than $\mathbf{A_{8\times8}} \times \mathbf{B_{8\times16}}$, the execution time with TPL will be shorter than that without TPL. We thus set the threshold of the matrix multiplication of a square matrix and a rectangular matrix at $\mathbf{A_{8\times8}} \times \mathbf{B_{8\times16}}$. Similarly to the matrix multiplication, the threshold of the other matrix operations can be set according to the experimental results. The threshold of the matrix addition of two square matrices in Table 3 is set to $\mathbf{A_{32\times32}} + \mathbf{B_{32\times32}}$ and the threshold of the matrix subtraction of two square matrices is the same as that of the matrix addition due to similar behavior; the threshold of the matrix divided by a constant in Table 4 is set to $\mathbf{A_{32\times32}}/c$; the threshold of the matrix hyperbolic tangent in Table 5 is set to $tanh(\mathbf{A_{32\times32}})$. The determined thresholds are used to decide whether the computation is complex or simple for the proposed CA procedure as addressed in next subsection.

Table 1: Execution time results of matrix multiplication

Matrix Operation	$\mathbf{A_{8\times8}} \times \mathbf{B_{8\times8}}$	$\mathbf{A_{16\times16}} \times \mathbf{B_{16\times16}}$	$\mathbf{A_{32\times32}} \times \mathbf{B_{32\times32}}$	$\mathbf{A_{64\times64}} \times \mathbf{B_{64\times64}}$
With TPL	0.028 ms	0.055 ms	0.321 ms	2.499 ms
Without TPL	0.016 ms	0.126 ms	0.975 ms	7.885 ms

Table 2: Execution time results of matrix multiplication

Matrix Operation	$\mathbf{A_{8\times8}} \times \mathbf{B_{8\times16}}$	$\mathbf{A_{8\times8}} \times \mathbf{B_{8\times32}}$	$\mathbf{A_{8\times8}} \times \mathbf{B_{8\times1024}}$	$\mathbf{A_{8\times8}} \times \mathbf{B_{8\times4096}}$
With TPL	0.027 ms	0.039 ms	0.764 ms	2.847 ms
Without TPL	0.035 ms	0.069 ms	2.267 ms	9.137 ms

Table 3: Execution time results of matrix addition

Matrix Operation	$\mathbf{A_{8\times8}} + \mathbf{B_{8\times8}}$	$\mathbf{A_{16\times16}} + \mathbf{B_{16\times16}}$	$\mathbf{A_{32\times32}} + \mathbf{B_{32\times32}}$	$\mathbf{A_{64\times64}} + \mathbf{B_{64\times64}}$
With TPL	0.0089 ms	0.0133 ms	0.0231 ms	0.0491 ms
Without TPL	0.0016 ms	0.0063 ms	0.0245 ms	0.0985 ms

Table 4: Execution time results of matrix divided by a constant

Matrix Operation	$\mathbf{A_{8\times8}}/c$	$\mathbf{A_{16\times16}}/c$	$\mathbf{A_{32\times32}}/c$	$\mathbf{A_{64\times64}}/c$
With TPL	0.0089 ms	0.0125 ms	0.0210 ms	0.0426 ms
Without TPL	0.0013 ms	0.0052 ms	0.0197 ms	0.0779 ms

Table 5: Execution time results of matrix hyperbolic tangent

Matrix Operation	$tanh$ $(\mathbf{A_{8\times8}})$	$tanh$ $(\mathbf{A_{16\times16}})$	$tanh$ $(\mathbf{A_{32\times32}})$	$tanh$ $(\mathbf{A_{64\times64}})$
With TPL	0.0136 ms	0.0220 ms	0.0664 ms	0.1409 ms
Without TPL	0.0092 ms	0.0303 ms	0.1144 ms	0.4342 ms

B. CA TPL Utilization Procedure for the FastICA Algorithm

Although TPL can dynamically adjust the degree of parallelism on all available cores, the TPL suffers from overhead when low computations are involved. Based on the threshold-testing results, the CA TPL utilization procedure [13] can be designed to use TPL efficiently to reduce the execution times of the FastICA algorithm. The proposed TPL utilization procedure divides the matrix operations into simple computation and complex computation. The complex computation uses TPL, and the simple computation does not. The overall goal is to reduce the execution time of complex computation while avoiding the overhead of TPL for the execution time of simple computation. To this end, we design a CA TPL utilization procedure for the FastICA algorithm. The CA TPL utilization procedure classifies each matrix operation into a complex or simple computation and then decides whether TPL is applied to the matrix operation. The FastICA algorithm includes the steps of centering (abbreviated as C), whitening (abbreviated as W), and the fixed-point algorithm (abbreviated as F). In the FastICA algorithm, centering is the first step. The centering step needs to calculate $E\{x_i\}$ and subtract $E\{x_i\}$ from $x_i(j)$ to generate $\bar{x}_i(j)$ with zero mean. The flow chart of the centering step using CA TPL is shown in Fig. 1. In Fig. 1, C.2, C.5, C.9 are used to detect the matrix dimension size to determine whether the TPL should be used.

After the centering step, the whitening process is performed. The whitening step needs to execute the eigenvalue decomposition (EVD). In this work, the eigenvalue decomposition is implemented with the cyclic Jacobi method [29] to obtain the eigenvalues and eigenvectors of a symmetric matrix. The cyclic Jacobi method applies a sequence of Jacobi rotations to the right side and the left side of the symmetric matrix. After obtaining the eigenvalue matrix, **D,** and eigenvector matrix, **E**, we can use (10) to generate the whitening signal matrix, **Z,** and finish the whitening step. Fig. 2 shows the flow chart of the whitening step using CA TPL. In Fig. 2, W.2, W.5, W.9, W.13 are used to detect the matrix dimension size to determine whether the TPL should be used. For the fixed-point algorithm, we first initialize a vector $\mathbf{w}_i$ with a unit norm for *i=1, 2, 3, ..., n*. Second, we compute $\mathbf{w}_i^+ = E\{\tilde{\mathbf{z}}[g(\mathbf{w}_i^T\tilde{\mathbf{z}})]\} - E\{g'(\mathbf{w}_i^T\tilde{\mathbf{z}})\}\mathbf{w}_i$. In Step 3, $[\mathbf{W}^+(\mathbf{W}^+)^T]^{-1/2}$ can be obtained by $\mathbf{E}\mathbf{D}^{-1/2}\mathbf{E}^T$, where **D** and **E** are obtained by the eigenvalue decomposition of $\mathbf{W}^+(\mathbf{W}^+)^T$. Once the weight matrix is converged, the fixed-point algorithm will stop. The flow chart of the fixed-point algorithm using CA TPL is shown in Fig. 3. In Fig. 3, F.3, F.6, F.10 are used to detect the matrix dimension size to determine whether the TPL should be used.

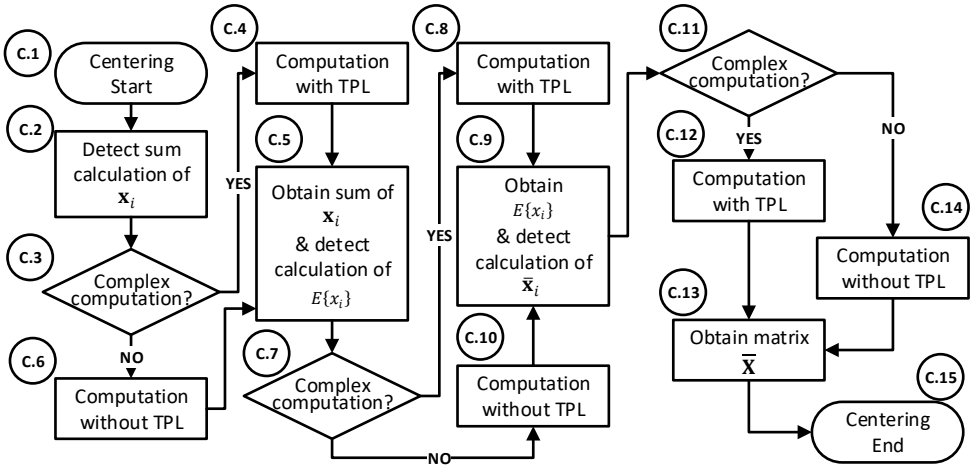


Fig. 1: Flow chart of the centering step.

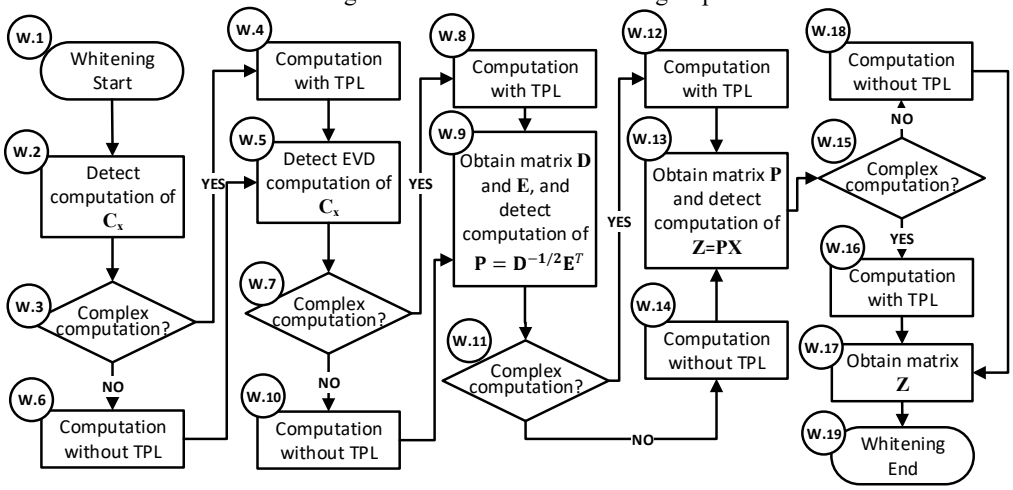


Fig. 2: Flow chart of the whitening step.

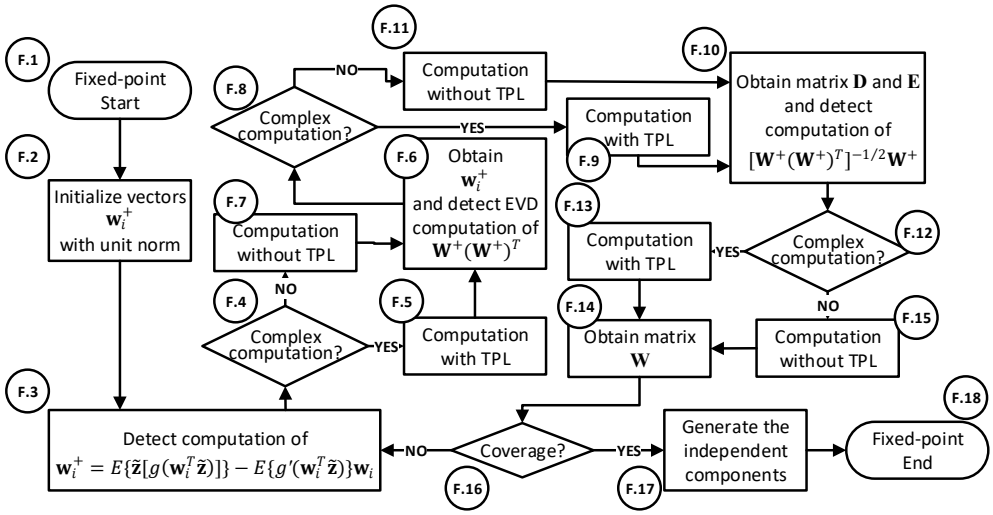


Fig. 3: Flow chart of the fixed-point algorithm.

In summary, the proposed computation-aware TPL utilization procedure [13] for the FastICA processing is composed of Fig. 1, Fig. 2, and Fig. 3. The next section presents the results of decomposing simulated and actual signals.

IV. Simulation and Comparison Results

A. Functional Validation

This subsection evaluates the proposed CA TPL utilization procedure applied to the FastICA algorithm. The evaluation is conducted on 8-channel artificially mixed source signals with a data length of 1024 as shown in Fig. 4. The mixed signals are processed by the proposed CA TPL utilization procedure of the FastICA algorithm. Fig. 5 shows the separation results. As can be seen, the mixed signals are well separated by the proposed CA TPL utilization procedure. The average absolute correlation coefficient between the separation results and the source signals is 0.9975. Therefore, the function of the proposed CA TPL utilization procedure can be validated.

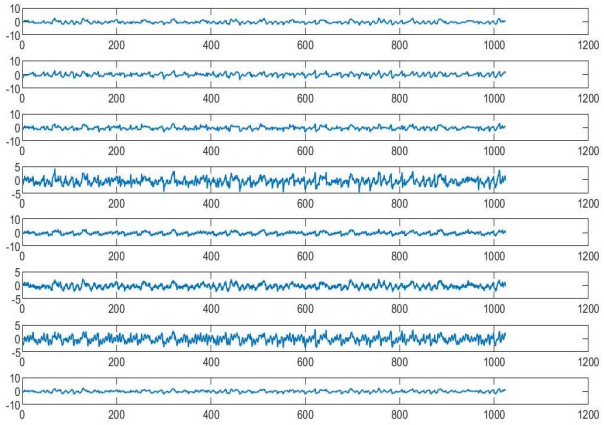
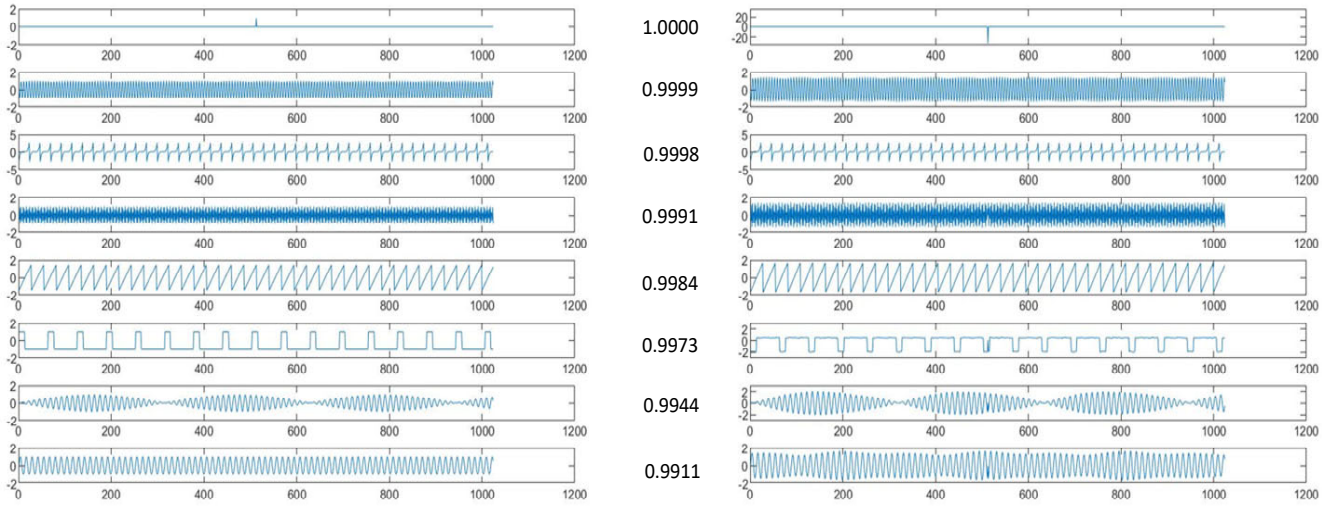


Fig. 4: 8-channel mixed signals.



(a) (b)

Fig. 5: Comparison results of (a) the source signals and (b) the separation results using the proposed CA TPL utilization procedure.

B. Simulation Results

In this subsection, we tested the proposed CA TPL utilization procedure on three datasets on a PC featuring Intel Core 2 Quad Q8200 to obtain the execution time of the FastICA algorithm. Besides the proposed procedure program, two referenced programs are implemented. One referenced program is implemented without TPL. The other program parallelizes the computation with TPL to all matrix operations without distinguishing the simple computation and complex computation. The same datasets are processed by three programs. Therefore, we can do a fair comparison among the three implementations.

The first dataset contains 8-channel artificial mixed signals with a data length of 921,600 samples. Table 6 shows the execution times of each step of the FastICA algorithm. As can be seen, the execution time is mainly contributed by the processing of the fixed-point algorithm. This is because the fixed-point algorithm dominates the computational complexity of the FastICA algorithm. The similar effect can be observed in the execution times of the other two datasets. For the first dataset, the proposed CA TPL utilization procedure can reduce 34.88% execution times, compared to the program without TPL. Compared to the fully parallelized program with TPL, the proposed CA TPL utilization procedure can reduce the execution time by 10.04%.

The second dataset contains 32-channel artificial mixed signals with a data length of 2,048 samples. Table 7 shows the execution times. For the second dataset, the proposed CA TPL utilization procedure can reduce the execution time by 43.01%, compared to the program without TPL. However, compared to the fully parallelized program with TPL, the execution time of the proposed efficient TPL utilization procedure is only reduced by 0.93%. This is because the most matrix operations are classified as a complex computation since the channel of the dataset is up to 32. Nevertheless, the proposed CA TPL

utilization procedure can still achieve the goal of performance improvement compared to the program without TPL.

The third dataset contains 12-channel real EEG signals with a data length of 276,360 samples. Table 8 shows the execution time results. For the third dataset, the proposed CA TPL utilization procedure can reduce the execution time by 48.27%, compared to the program without TPL. Compared to the fully parallelized program with TPL, the proposed CA TPL utilization procedure can reduce the execution time by 15.12%.

Tables 9, 10, and 11 show the number of TPL utilizations for the above three datasets. Obviously, the program without TPL has zero utilization, and the parallelized program with TPL has the highest number of utilizations. The proposed CA TPL utilization procedure for the FastICA algorithm will not use TPL for the simple computation to avoid the overhead. Thus, the number of TPL utilizations of the proposed CA TPL utilization procedure is less than that with the fully parallelized program with TPL and higher than that of the program without TPL. According to the above experiment results, we can find that the proposed CA TPL utilization procedure of the FastICA algorithm can improve the execution time.

Table 6: Execution time results with the first dataset

FastICA Steps	C	W	F	Total
Without TPL	309 ms	4827 ms	68709 ms	73845 ms (100%)
With TPL	307 ms	2784 ms	50365 ms	53456 ms (72.39%)
This work [13]	306 ms	2586 ms	45195 ms	48087 ms (65.12%)

Table 7: Execution time results with the second dataset

FastICA Steps	C	W	F	Total
Without TPL	3 ms	8663 ms	234270 ms	242936 ms (100%)
With TPL	8 ms	6899 ms	132836 ms	139743 ms (57.52%)
This work [13]	3 ms	6919 ms	131525 ms	138447 ms (56.99%)

Table 8: Execution time results with the third dataset

FastICA Steps	C	W	F	Total
Without TPL	161 ms	3540 ms	117065 ms	120766 ms (100%)
With TPL	171 ms	1504 ms	71921 ms	73599 ms (60.94%)
This work [13]	146 ms	1472 ms	60854 ms	62472 ms (51.73%)

Table 9: Number of TPL utilizations of the first dataset

FastICA Steps	C	W	F	Total
Without TPL	0	0	0	0 (0%)
With TPL	2	6	254	262 (100%)
This work [13]	2	2	67	71 (27.10%)

Table 10: Number of TPL utilizations of the second dataset

FastICA Steps	C	W	F	Total
Without TPL	0	0	0	0 (0%)
With TPL	2	6	312	320 (100%)
This work [13]	2	3	144	149 (46.56%)

Table 11: Number of TPL utilizations of the third dataset

FastICA Steps	C	W	F	Total
Without TPL	0	0	0	0 (0%)
With TPL	2	6	599	607 (100%)
This work [13]	2	2	157	161 (26.52%)

V. Conclusion

This study proposed a computation-aware TPL utilization procedure for the FastICA algorithm. The proposed TPL utilization procedure applies TPL on complex computation but not on simple computation. It can achieve less execution time on a multi-core CPU. Compared to the program without TPL, the study results showed that the execution time of the FastICA decomposition applied to 8-channel artificial mixed signals with a data length of 921,600 was reduced by 34.88%. Compared to the fully parallelized program with TPL, the proposed procedure could reduce the execution time by 10.04%. The execution time of the FastICA decomposition applied to 32-channel artificial mixed signals with a data length of 2,048 could be reduced by 43.01%, compared to the program without TPL. Compared to the fully parallelized program with TPL, the proposed procedure reduced the execution time by 0.93%. The execution time of the FastICA decomposition applied to 12-channel EEG signals with a data length of 276,360 was reduced by 48.27% compared to the program without TPL. Compared to the parallelized program with TPL, the proposed procedure reduced the execution time by 15.12%. In the near future, we plan to further improve the performance by heterogeneous computing hardware resources.

REFERENCES

[1] A. J. Bell and T. J. Sejnowski, "An information maximization approach to blind separation and blind deconvolution," Neural Computation, vol. 7, pp. 1129-1159, 1995.

[2] A. Hyvärinen and E. Oja, "A fast fixed-point algorithm for independent component analysis," Neural Computation, vol. 9, pp. 1483–1492, 1997.

[3] S. Choi, A. Cichocki and S. Amari, "Flexible independent component analysis," J. VLSI Signal Process., vol. 26, nos. 1-2, pp. 25-38, 2000.

[4] A. Hyvärinen, J. Karhunen, and E. Oja, Independent Component Analysis. New York: Wiley, 2001.

[5] A. Kachenoura , L. Albera , L. Senhadji and P. Comon "ICA: A potential tool for BCI systems," IEEE Signal Processing Magazine, vol. 25, no. 1, pp. 57-68, Jan. 2008.

[6] L. D. Van, D. Y. Wu, and C. S. Chen, "Energy-efficient FastICA Implementation for biomedical signal separation," IEEE Transactions on Neural Networks, vol. 22, no. 11, pp. 1809-1822, Nov. 2011.

[7] L. D. Van, P. Y. Huang, and T. C. Lu, "Cost-effective and variable-channel FastICA hardware architecture and implementation for EEG signal processing," Journal of Signal Processing Systems, vol. 82, issue 1, pp. 91-113, Jan. 2016.

[8] L. D. Van, T. C. Lu, T. P. Jung, and J. F. Wang, "Hardware-oriented memory-limited online FastICA algorithm and hardware architecture for signal separation," in Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), May 2019, pp. 1438-1422, Brighton, UK.

[9] M. Sajjad, M. Z. Yusoff, N. Yahya, and A. S. Haider, "An efficient VLSI architecture for FastICA by using the algebraic Jacobi method for EVD," *IEEE Access*, vol. 9, pp. 58287-58305, 2021.

[10] D. B. Keith, C. C. Hoge, R. M. Frank, and A. D. Malony, "Parallel ICA methods for EEG neuroimaging," in Proc. 20th IEEE International Parallel & Distributed Processing Symposium, Apr. 2006.

[11] R. Ramalho, P. Tomás, and L. Sousa, "Efficient independent component analysis on a GPU," in Proc. 10th IEEE International Conference on Computer and Information Technology, Jun. 2010, pp. 1128-1133.

[12] D. Brandt, Investigation of GPGPU for Use in Processing of EEG in Real-Time, Master Thesis, Rochester Institute of Technology, USA, 2010.

[13] Sing-Jia Tzeng, Efficient TPL Utilization Procedure for Paralleling FastICA Algorithm on Multi-Core CPU, Master Thesis, National Chiao Tung University, Taiwan, 2014. (Advisor: Lan-Da Van)

[14] M. Plauth, F. Feinbube, P. Tröger, and A. Polze, "FastICA on modern GPU architectures," in Proc. 15th IEEE International Conference on Parallel and Distributed Computing, Applications and Technologies, Dec. 2014, pp. 69-75.

[15] M. Fang, J. Fang, and W. Zhang, "Efficient and portable parallel framework for hyperspectral image dimensionality reduction on heterogeneous platforms," Journal of Applied Remote Sensing, vol. 11(1), Jan.-Mar. 2017.

[16] G. Benko and Z. Juhasz, "GPU implementation of the FastICA algorithm," in Proc. 42nd IEEE International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), May 2019, pp. 196-199.

[17] F. Carrizosa-Corral, A. Vazquez-Cervantes, J. R. Montes, T. Hernandez-Diaz, J. C. Solano Vargas, L. Barriga-Rodriguez, J. A. Soto-Cajiga, H. Jimenez-Hernandez, "FPGA-SoC implementation of an ICA-based background subtraction method," *International Journal of Circuit Theory and Applications*, vol. 46, pp. 1703-1722, Apr. 2018.

[18] K. P. Murphy, Machine Learning: A Probabilistic Perspective. MIT press, 2012.

[19] E. Bingham, S. Kaski, J. Laaksonen, and J. Lampinen, (Eds.), Advances in independent component analysis and learning machines. Academic Press, 2015.

[20] OpenMP.org. [Online] Available: http://www.openmp.org

[21] W. Gropp, E. Lusk and A. Skjellum, Using MPI: Portable Parallel Programming With Message-Passing Interface. MIT Press, 1994

[22] CUDA Zone. [Online] Available: https://developer.nvidia.com/category/zone /cuda-zone

[23] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," Computing in Science & Engineering, vol. 12, issue 3, pp. 66-72, May-Jun. 2010.

[24] Khronos OpenCL Working Group: OpenCL Specification. [Online]. Available: http://www.khronos.org/registry/cl/specs/opencl-2.0.pdf

[25] D. Leijen, W. Schulte and S. Burckhardt "The design of a task parallel library," ACM SIGPLAN Notices, vol. 44, issue 10, pp 227–242, Oct. 2009.

[26] Microsoft: TPL. [Online] Available: https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-parallel-library-tpl

[27] Microsoft: C# Programming. [Online] Available: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/

[28] Microsoft: Stopwatch Class. [Online] Available: http://msdn.microsoft.com/en-us/library/system.diagnostics.stopwatch(v=vs.100).ASPX

[29] G. H. Golub and C. F. Van Loan, Matrix Computation. Johns Hopkins University Press, 1996.