A GPU Implementation of FastICA in Audio Applications for Small Number of Components

Stefan Kanan*
kanan.stefan@students.finki.ukim.mk
Ss. Cyril and Methodius
Skopje, Macedonia

Marjan Gusev
marjan.gushev@finki.ukim.mk
Ss. Cyril and Methodius
Skopje, Macedonia

Vladimir Zdraveski
vladimir.zdraveski@finki.ukim.mk
Ss. Cyril and Methodius
Skopje, Macedonia

ABSTRACT

Extracting independent components from audio data has plenty of uses in biology, music, communication and media and in many other fields. The FastICA algorithm is a relatively fast and simple algorithm, that assumes the original sources to be nongaussian and works by lowering the gaussianity of the mixed sources. Yet as the number of samples increase so does the time required for its execution. While one solution would be to simply use just a subset of the samples, in this paper we look at the possibility of extending the FastICA algorithm to the GPU. While similar efforts have been pursued in the past, we deal with data of relatively few components, as would be more common when dealing with audio data. We implement a fully GPU FastICA as well as a CPU-GPU hybrid algorithm, both based on the CUDA platform and compare them with the CPU version. Our results indicate that for large samples the CPU-GPU hybrid and the GPU algorithms perform better than their CPU counterpart.

CCS CONCEPTS

• **Mathematics of computing → Computations on matrices**; *Distribution functions*; • **Computing methodologies → Parallel algorithms**.

KEYWORDS

Independent Component Analysis, CUDA, Parallel Programming, Negentropy, Nongaussianity

ACM Reference Format:
Stefan Kanan, Marjan Gusev, and Vladimir Zdraveski. 2019. A GPU Implementation of FastICA in Audio Applications for Small Number of Components. In *Proceedings of Sofia '19: 9th Balkan Conference in Informatics (Sofia '19).* ACM, New York, NY, USA, 4 pages.

1 INTRODUCTION

Encountered with a noisy environment at a party, humans seem to be innately capable at muffling extraneous noises and focusing on those relevant to them. The problem of separating signals in regards to sound is called the 'Cocktail Party' problem. The field tasked with finding a way to recreate this phenomena in the brain and apply it to other fields is called blind source separation (BSS).

This has its uses in transcribing music heard from an audio source, making better robotic translators [9], it can help develop better and more advanced hearing aids [3] and even help animal biologists [3], it has its uses in non-audio related problems too, such as helping signal diagnosing in electrocardiogram (ecg) [1] and electroencephalogram (eeg) [1] [9] [10].

In our paper we propose a parallel version of the FastICA algorithm (an independent component analysis algorithm) in which audio signals are reclaimed from several recordings. We imagine this to be useful in several scenarios, such as: obtaining a clean interview under noisy environment, combining different audio recordings from a concert and refining the quality, transcription and annotation of music, removing feedback loops from video conferences and etc.

Our paper is organized as follows. In Section 2 we refer the reader to some work that may be related to ours, in Section 3 we review the theory behind Independent Component Analysis, in Section 4 we discuss the FastICA algorithm and describe our CUDA implementation. Further in Section 5 we discuss our datasets on which we test our FastICA implementations, and present our findings. Finally in Section 6 we consider some improvements and directions in which can be further explored, and give a conclusion in Section 7.

2 RELATED WORK

A CPU parallelized implementations of the INFOMAX and the deflation type FastICA are offered in [6] with an accent on large scale datasets that usually are impractical for use on normal implementations. It is implemented in EEGLAB (a toolset for neuroscience) and reportedly provides an impressive boost to performances in the FastICA case.

However using the deflation FastICA means that signals are processed sequentially, Hyvarian, thus there's an inherent bottleneck that makes it unsuitable for use in a GPU. Another approach is called Symmetric FastICA which works exclusively with matrices, and so is more natural for use in a multi-core GPU.

Finally [13] presents a Symmetric FastICA implementation on the CUDA platform and in doing so consider several different approaches (A CPU Lapack version, a GPU-CPU hybrid version and a fully parallelized version). In the end they find the fully parallelized FastICA to have a speedup of 55x for 256 signals[13]. In our paper we make use of their finding by implementing our own version of the GPU FastICA algorithm and testing it on audio signals.

Sofia '19, September 26–28, 2019, Sofia, Bulgaria

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

3 THE THEORETICAL BACKGROUND OF INDEPENDENT COMPONENT ANALYSIS

Suppose a scenario in which we have n musicians playing on their instruments. At the same time they are also recorded by as many microphones placed around the room. Each of the microphones ends up recording all of the instruments, however as the sounds travel, they clash with one another and bounce off the geometry of the hall prior to reaching the recording devices. Because of this, each microphone will receive a distorted concoction of all the separate signals. This can be written in the matrix form: [11] [4] [5]

$$X = As \tag{1}$$

. Where s is a matrix of form nxm whose m columns represent the time frames and whose n rows represent the unadulterated signals. We assume that the initial signals are statistically independent[1]. A is called the mixing matrix, a nxn matrix containing the coefficients by which the original signals are mixed prior to reaching the recording devices. X is the resulting matrix, each row of X contains all of the signals, each either attenuated or amplified.

The goal of independent component analysis now, is to somehow find each thread of the signal and then unravel it to its original form. This is made more difficult however by the fact that we only have the matrix X at our disposal (hence the title of Blind Source Separation). Yet, because of the central limit theorem we know that the mixed signals will exhibit a more Gaussian distribution [4], by assuming that the primary signals have a nongaussian distribution [1], we can try to reconstruct these signals by heightening the non-gaussianity of our distribution in each row of our matrix [9]. We may be inclined to believe that the original signals will too have a Gaussian distribution, however this is often not the case[4]. For instance audio signals have often been found to have a supergaussian (peaked edge) distribution[14] [2].

Should we be successful in our attempt of finding the different source signals, we will have found the inverse of the mixing matrix A, thus gaining the ability to estimate the original signals.

$$\hat{s} = A^{-1}X \tag{2}$$

We will denote the inverse of A as $W = A^{-1}$. A Gaussian random variable has the biggest entropy amongst the random variables with the same variance[4]. Therefore entropy provides a good measure of nongaussianity. We can further employ the measure of negentropy [4]

$$J(y) = H(y_{gaussian}) - H(y) \tag{3}$$

. Which is always non-negative and zero for random variables of a Gaussian distribution[4] [11]. Where $y = \hat{s}$ and $y_{gaussian}$ is a random Gaussian variable with the same correlation as y. $H(y)$ is the entropy defined as

$$H(y) = -\int p_y(\eta) \log p_y(y) d\eta \tag{4}$$

. We assume y to be a random variable of zero mean and a unit variance[4]. We can approximate $J(y)$ with higher-order cumulants, the approximation in the end, has the form of [4]

$$J(y) \propto [E\{G(y)\} - E\{G(v)\}]^2 \tag{5}$$

. Where $E\{X\}$ is the expectation and G is a non-quadratic function, and v is a normal Gaussian variable.

4 DISCUSSION OF THE FASTICA ALGORITHM

FastICA uses the Newton approximation method in combination with our negentropy measure in order to find a suitable unmixing matrix W that minimizes the gaussianity of our signals. Let X be our matrix containing each of the n mixed signals in its rows. Because FastICA expects the distribution to have mean 0 [4], we first normalize the rows by subtracting their mean [4]. Once independent we can return their mean. Prior to starting the FastICA algorithm, it is encouraged to first whiten the data [9] [1]. Whitening both makes the data uncorrelated and its variance equal 1[4]. Another property of a whitened matrix is that it becomes orthogonal [4]. This makes the process easier as it means that we only have to search for an orthogonal matrix thus narrowing the searchspace [4]. As stated in the previous section, the function $G(y)$ can be chosen to be any non-quadratic function. [4] offers several non-quadratic functions reported to have good convergence properties. The function used in our implementation as well as its derivatives are given below:

$$\begin{aligned} G(y) &= \frac{\log \cosh a_1 y}{a_1} \\ g(y) &= \tanh a_1 y \\ g'(y) &= a_1(1 - \tanh a_1 y^2) \end{aligned} \tag{6}$$

Usually $a_1 = 1$. Let $y = W^T X$ be our approximation of the original signals. We find the optima of the negentropy measure using a Lagrangian of $E\{G(W^T X)\}$.

$$E\{Xg(W^T X)\} + \beta W = 0 \tag{7}$$

Finding the derivative of this function, we get:

$$\frac{\partial F}{\partial W} = E\{XX^T\}E\{g'(W^T X)\} + \beta I = E\{g'(W^T X)\}I + \beta I \tag{8}$$

Because whitening X makes it orthogonal, $XX^T = I$. Using a Newton approximation:

$$W = W - \frac{[E\{Xg(W^T X)\} + \beta W]}{[E\{g'(W^T X)\} + \beta]} = E\{Xg(W^T X)\} - E\{Xg'(W^T X)W\} \tag{9}$$

. We get a new direction for W. As stated in [4] in order to orthogonalize W, we can use

$$\begin{aligned} 1.W &= \frac{W}{\|W\|} \\ 2.W &= \frac{3}{2}W - \frac{1}{2}WW^T W \end{aligned} \tag{10}$$

where $\|W\|$ is the matrix norm. In our case that is either the absolute max of the columns or rows. This process continues until W becomes orthogonal, or rather until $WW^T = \hat{I}$ for some threshold τ [4] [13]. Matrix orthogonalization can also be done using

$$W = W(W^T W)^{-\frac{1}{2}} \tag{11}$$

. Though the former has been found to have better performances on the GPU [13]. The FastICA algorithm consists of finding a new W using equation (9), and then orthogonalizing it using either (10) or (11). FastICA converges when $W_{i-1}W_i^T \approx \hat{I}$ for some threshold τ.

Intitialize random matrix W of size nxn;
while $WW_{n-1}^T ! \approx I$ **do**
  $W = W(W^TW)^{-\frac{1}{2}}$;
  $W_{n-1} = W$;
  $W = E\{Xg(W^TX)\} - E\{Xg'(W^TX)W\}$
end

Algorithm 1: Serial FastICA. Calculates the square root of the matrix W directly.

Intitialize random matrix W of size nxn;
toGPU(W);
while $WW_{n-1}^T ! \approx I$ **do**
  while $WW^T ! \approx I$ **do**
    $W = \frac{W}{\|W\|}$;
    $W = \frac{3}{2}W - \frac{1}{2}WW^TW$
  end
  $W_{n-1} = W$;
  $W = E\{Xg(W^TX)\} - E\{Xg'(W^TX)W\}$
end

Algorithm 2: GPU version of the FastICA algorithm. Approximates the square of the matrix W.

5 DATASET AND RESULTS

Our FastICA implementations were tested on an Intel Xeon CPU (2.30 GHZ) with 8GB of RAM and an NVIDIA TESLA K80 card for all the GPU calculations. Our data consisted of 11 tracks in the 'WAV' audio format at a 44100 Hz samplerate and at 32 bit depth. The tracks were then combined into the data matrix s, and mixed using random gaussian mixing matrix A of dimensions $11x11$. Finally the mixed matrix $X = As$ was created, this matrix is of the form $11xM$. 5 minutes of this track correspond to about 13530790 frames.

We calculated the speedup of our GPU version by comparing the performances with the serial version as

$$speedup = \frac{Serial}{V} \tag{12}$$

. Where $Serial$ is average times of execution for the serial algorithm and V is the average time of execution for either the GPU or hybrid algorithm. The GPU version of this algorithm described in Algorithm 2 is based on the CUDA platform. The platform enables us to employ the vast resources of our GPU for any computational task that we may have, cutting the computational time in a very significant way. We use the python PyCuda [8] library, a wrapper for the CUDA API. We also use both the numpy and scipy libraries [12] for the serial parts of our implementations. [13] reports a $55x$ speed increase compared to the serial version and a $13x$ speed increase compared to the serial version using the optimized MLK library when testing for 256 components, however this amount of independent components would be an unrealistic assumption for audio data. By making the GPU deal with small matrices repeatedly, it might wane any purported speedup because of the large overhead. To test this we also developed a CPU-GPU hybrid version -described in Algorithm 3- which differs from the GPU version by using the CPU for orthogonalizing W. We compared these algorithms with a serial implementation of FastICA, which is described in Algorithm 1.

Intitialize random matrix W of size nxn;
toGPU(X);
while $WW_{n-1}^T ! \approx I$ **do**
  $W = W(W^TW)^{-\frac{1}{2}}$;
  $W_{n-1} = W$;
  toGPU(W);
  $g_1 = E\{Xg(W^TX)\}$;
  $g_2 = E\{Xg'(W^TX)W\}$;
  toCPU(g_1);
  toCPU(g_2);
  $W = g_1 - g_2$
end

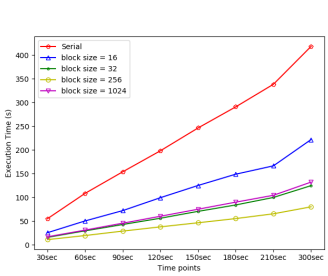
Algorithm 3: Hybrid FastICA. Calculates the square root of the matrix W directly.

Each of the algorithms were let to run for 100 iterations, in the case of the GPU algorithm we set the orthogonalization approximation converge for $\tau = 0.001$.

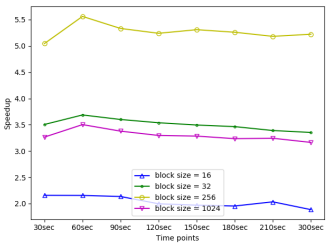
Figure a and b show the runtime performance and speedup of the GPU algorithm compared to the CPU implementation. Moreover Figure d and e show the same for the Hybrid algorithm. We can see that the performance of these algorithms is nearly identical, with the speedup being rather constant. Figure c shows the total time for every multiplication containing X (our data) as well as well as the total time it took for the non-linear tanh function to finish. Both of these functions are shared in the GPU and Hybrid implementations. We can see benefit in using our GPU multiplication function compared to their CPU counterparts for thread blocks above 16 (Only thread blocks for 16 and 256 shown for clarity). Moreover while it took the serial version a long time to calculate all non-linear functions, it took mere milliseconds for the GPU algorithms to do the same job, therefore giving us a massive performance gain. Figure f shows the total time it took for memory to be transferred from the CPU to the GPU and vice versa. Some other runtime metrics include convergence checks (500ms for GPU, 0.3ms for CPU and Hybrid) and orthogonlaization (1.5s for GPU, 0.09 for CPU and Hybrid) whose plots were omitted for brevity.

6 FURTHER WORK

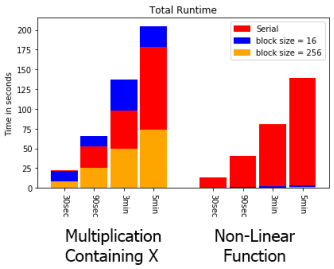
While we showed the effectiveness of the algorithm it would be interesting to see how well it performs under real settings as well as the quality of the extracted tracks. Because FastICA is order invariant it would not be possible for an ex. communication application to figure out where each of the signals originated from. Such an application would therefore be in need of a framework in which the independent signals are first extracted and then classified before



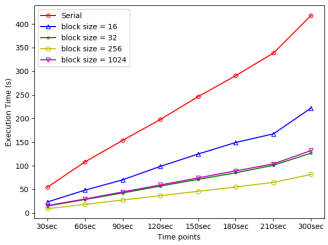
(a) Gpu vs Serial Runtime



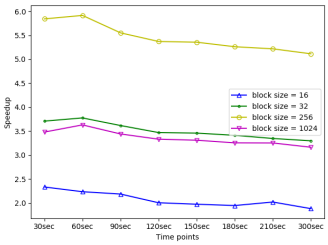
(b) Gpu Speedup



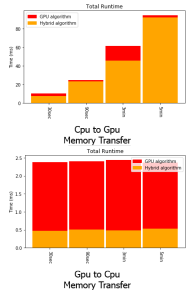
(c) Runtime Profiling



(d) Hybrid vs Serial Runtime



(e) Hybrid speedup



(f) Memory Transfer Profiling

being sent into their corresponding destinations. One problem with the ICA algorithms is that they require multiple sources, and while this is common for some applications such as in EEG and ECG which naturally come with multiple observations. We often don't get that luxury with audio data, and so audio tracks may often need to be aligned or preprocessed in some way. More than that audio data often comes in fewer sources than there are components to be extracted (ex. a phone camera footage, a lone microphone listening in the wild). A group of algorithms more suitable in these conditions are the Independent Subspace Analysis algorithms [7]. Audio data would first need to be converted to the frequency domain and then using the ISA algorithm have its independent components be extracted. It would be interesting to see whether a CUDA ISA algorithm would offer any performance gains.

7 CONCLUSION

FastICA is an independent component analysis algorithm that depends on matrix operations, thus is a good candidate for GPU parallelization. While GPU parallelizations have been done in the past, in this paper we try to see if there is any benefit in using the FastICA algorithm for datasets with few components. Specifically we work with extracting components from audio data. A GPU and a Hybrid implementation of the FastICA algorithm were made, and their performance was compared to that of the serial version. Our results show merit in using the parallel version of the algorithm with a near constant $5x$ speedup, though shows almost no differnce between the GPU and Hybrid algorithms.

REFERENCES

[1] Erkki Oja Aapo Hyvärinen, Juha Karhunen. 2002. What is Independent Component Analysis? *Independent Component Analysis* (May 2002). https://doi.org/10.1002/0471221317.ch7

[2] Akhil Arora, Rajesh Mehra, and Naveen Dubey. 2015. Performance Analysis Of Sub And Super-Gaussian Blind Audio Source Separation. (2015), 5.

[3] Mark A. Bee and Christophe Micheyl. 2008. The Cocktail Party Problem: What Is It? How Can It Be Solved? And Why Should Animal Behaviorists Study It? *Journal of comparative psychology (Washington, D.C. : 1983)* 122, 3 (Aug. 2008), 235–251. https://doi.org/10.1037/0735-7036.122.3.235

[4] Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. 2001. *Independent Component Analysis* (1 edition ed.). Wiley-Interscience, New York.

[5] E. S. Juan, I. Soto, P. Adasme, L. Cañete, F. Seguel, W. Gutierrez, C. Melendez, and A. D. Firoozabadi. 2018. Comparison between FastICA and InfoMax for the blind separation of audio signals. In *2018 11th International Symposium on Communication Systems, Networks Digital Signal Processing (CSNDSP)*. 1–4. https://doi.org/10.1109/CSNDSP.2018.8471822

[6] D. B. Keith, C. C. Hoge, R. M. Frank, and A. D. Malony. 2006. Parallel ICA methods for EEG neuroimaging. In *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*. 10 pp.–. https://doi.org/10.1109/IPDPS.2006.1639299

[7] Mohammed Khalil and Abdellah Adib. 2016. Multiple audio watermarking system based on CDMA. *International Journal of Information and Communication Technology* 9, 2 (2016), 160. https://doi.org/10.1504/IJICT.2016.078878

[8] Andreas Klöckner, Nicolas Pinto, Yunsup Lee, B. Catanzaro, Paul Ivanov, and Ahmed Fasih. 2012. PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation. *Parallel Comput.* 38, 3 (2012), 157–174. https://doi.org/10.1016/j.parco.2011.09.001

[9] Eleftherios Kofidis. 2016. Blind Source Separation: Fundamentals and Recent Advances (A Tutorial Overview Presented at SBrT-2001). *arXiv:1603.03089 [cs, math, stat]* (March 2016). http://arxiv.org/abs/1603.03089 arXiv: 1603.03089.

[10] Ahmad Mayeli, Vadim Zotev, Hazem Refai, and Jerzy Bodurka. 2016. Real-time EEG artifact correction during fMRI using ICA. *Journal of Neuroscience Methods* 274 (Dec. 2016), 27–37. https://doi.org/10.1016/j.jneumeth.2016.09.012

[11] S. Nikam and S. Deosarkar. 2016. Fast ICA based technique for non-invasive fetal ECG extraction. In *2016 Conference on Advances in Signal Processing (CASP)*. 60–65. https://doi.org/10.1109/CASP.2016.7746138

[12] Travis Oliphant. 2007. A guide to NumPy. http://www.numpy.org/

[13] R. Ramalho, P. Tomás, and L. Sousa. 2010. Efficient Independent Component Analysis on a GPU. In *2010 10th IEEE International Conference on Computer and Information Technology*. 1128–1133. https://doi.org/10.1109/CIT.2010.205

[14] Diego Renza, Sebastián Mendoza, and Dora M. Ballesteros L. 2019. High-uncertainty audio signal encryption based on the Collatz conjecture. *Journal of Information Security and Applications* 46 (June 2019), 62–69. https://doi.org/10.1016/j.jisa.2019.02.010