A CUDA IMPLEMENTATION OF INDEPENDENT COMPONENT ANALYSIS IN THE TIME-FREQUENCY DOMAIN

Radoslaw Mazur and Alfred Mertins

Institute for Signal Processing
University of Lübeck
23538 Lübeck, Germany
{mazur,mertins}@isip.uni-luebeck.de

ABSTRACT

For the blind separation of convolutive mixtures, a huge processing power is required. In this paper we propose a massive parallel implementation of the Independent Component Analysis in the time-frequency domain using the processing power of the current graphics adapters within the CUDA framework. The often used approach for solving the separation task is the transformation to the time-frequency domain where the convolution becomes a multiplication. This allows for the use of an instantaneous ICA algorithm independently in each frequency bin, which greatly reduces complexity. Besides algorithmic simplification, this approach also provides a very founded approach for parallelization. In this work, we propose an implementation using the CUDA framework, which provides an easy interface for the implementation of massive parallel algorithms. The new implementation allows for a speedup in the order of two magnitudes, as it will be shown on real-world examples.

1. INTRODUCTION

Blind Source Separation (BSS) is a method for restoring signals from observed mixtures. When neither the original signals nor the mixing system is known, only blind techniques can be used. In case of linear instantaneous mixtures and the assumption of statistically independent sources, Independent Component Analysis (ICA) may be employed for the separation [1, 2, 3].

When dealing with acoustic mixtures of speech this simple approach is not sufficient. The low speed of sound and reflections on objects results in soundwaves arriving multiple times with different lags. This convolutive mixing system can be described using FIR filters, but for realistic scenarios the length of these filters is usual up to several thousand taps. In this case the unmixing system consists again of FIR filters with at least the same length.

It is possible to calculate the unmixing filters directly in the time domain [4, 5]. However, this approach results in high computational cost and often shows difficulties with convergence, as the algorithm can get trapped in one of the many local minima of the objective function.

An often used approach is the transformation to the time-frequency domain where the convolution becomes a multiplication [6]. This allows the use of instantaneous methods in each frequency bin independently. However, if all frequency bins are separated independently, the discrete bins usually have different scalings, and they can be arbitrarily permuted. There exist different approaches for solving the permutation problem [7, 8, 9, 10, 11].

The correction of the different scaling in every frequency bin may be carried out using the postfilter method from [12]. This approach tries to recover the signals as they have been recorded at the microphones and thus accepts all filtering done by the mixing system without adding new distortions. A similar technique, the minimal distortion principle, has been proposed in [13]. New approaches as proposed in [14] and [15] solve the scaling problem with the aim of filter shortening or shaping.

In this paper we address the problem of the high computational cost of the time-frequency approach. The first step of the algorithm, the calculation of the time-frequency representation of the speech signals using the blockwise Short-Time Fourier Transform (STFT), is computationally unproblematic. But when using a gradient based approach for the ICA, as in [1], usually a few hundreds of iterations in every frequency bin are needed. The computational costs for this stage are usually too high to be carried out in real-time on a typical desktop or mobile CPU.

It is possible to reduce the number of iterations by careful initialization. One method is just to use the result of the neighboring bins as the starting point. In [16] the authors proposed a method for predicting the unmixing matrix in the next bin. With a good estimation the number of iterations can be greatly reduced. The major drawback of these approaches is the consequence of losing the independency of the individual bins which prohibits a parallel execution.

We propose to carry out these computations using a CUDA enabled graphics hardware [17]. These graphics adapters or the Tesla dedicated compute boards consist of up to 512 compute cores. Each of these cores is capable of performing all the necessary computations of the instantaneous ICA in a single frequency bin.

2. MODEL AND METHODS

2.1 BSS for Instantaneous Mixtures

In this section, we describe the instantaneous unmixing process that we use in each frequency bin of the convolutive scenario. The described gradient descent procedure needs a lot of computing power, as it has to be carried out at least a few hundred times in every discrete frequency bin.

The instantaneous mixing process of N sources into N observations can be modeled by an $N \times N$ matrix $\mathbf{A}$. With no measurement noise, a given source vector $\mathbf{s}(n) = [s_1(n), \ldots, s_N(n)]^T$ is transformed to an observation $\mathbf{x}(n) = [x_1(n), \ldots, x_N(n)]^T$ by

$$\mathbf{x}(n) = \mathbf{A} \cdot \mathbf{s}(n). \tag{1}$$

The separated signals $\mathbf{y}(n) = [y_1(n), \ldots, y_N(n)]^T$ may be obtained by a multiplication with an unmixing matrix $\mathbf{B}$:

$$\mathbf{y}(n) = \mathbf{B} \cdot \mathbf{x}(n). \tag{2}$$

The only sources of information for estimating $\mathbf{B}$ are the statistical properties of the observed signals $\mathbf{x}(n)$. The assumption of statistically independent sources only allows for a separation up to an unknown order and ambiguous scaling. Therefore the separation is considered successful when

$$\mathbf{BA} = \mathbf{D\Pi}, \tag{3}$$

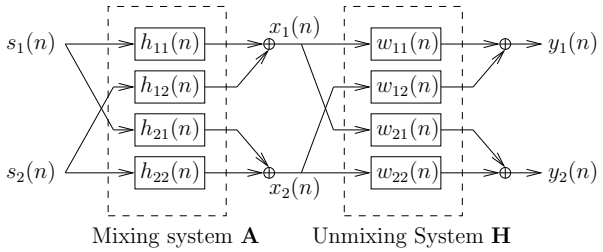

Figure 1: BSS model with two sources and sensors.

with $\mathbf{\Pi}$ being a permutation matrix and $\mathbf{D}$ an arbitrary diagonal matrix.

For the separation we use the Infomax algorithm with the natural gradient update [1]:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \Delta\mathbf{B}_k \tag{4}$$

with

$$\Delta\mathbf{B}_k = \mu_k(\mathbf{I} - E\left\{\mathbf{g}(\mathbf{y})\mathbf{y}^T\right\})\mathbf{B}_k \tag{5}$$

and $\mathbf{g}(\mathbf{y}) = (g_1(y_1), \dots g_n(y_n))$ being a component-wise vector function of nonlinear score functions $g_i(s_i) = -p_i'(s_i)/p_i(s_i)$, where $p_i(s_i)$ are the assumed source probability densities. These should be known or at least well approximated in order to achieve good separation performance [18]. For speech signals, a Laplacian distribution may be assumed [7, 8, 9]. In this case, the nonlinear score functions reduce to $g_i(s_i) = \mathrm{sgn}(s_i)$. For complex valued signals, with the assumption of spherically invariant distributions [19, 20], the score function reads $g_i(s_i) = \mathrm{sgn}(s_i)$ with

$$\mathrm{sgn}(s_i) = \frac{s_i}{|s_i|}. \tag{6}$$

2.2 Convolutive Mixtures

When dealing with real-world acoustic scenarios it is necessary to consider reverberation. The mixing system can be modeled by FIR filters of length L. Depending on the reverberation time and sampling rate, L can reach several thousand taps. The convolutive mixing model reads

$$\mathbf{x}(n) = \mathbf{H}(n) * \mathbf{s}(n) = \sum_{l=0}^{L-1} \mathbf{H}(l)\mathbf{s}(n-l) \tag{7}$$

where $\mathbf{H}(n)$ is a sequence of $N \times N$ matrices containing the impulse responses of the mixing channels. For the separation we use FIR filters of length M and obtain

$$\mathbf{y}(n) = \mathbf{W}(n) * \mathbf{x}(n) = \sum_{l=0}^{M-1} \mathbf{W}(l)\mathbf{x}(n-l) \tag{8}$$

with $\mathbf{W}(n)$ containing the unmixing coefficients. Figure 1 shows the scenario for two sources and sensors.

Using the short-time Fourier transform (STFT), the signals can be transformed to the time-frequency domain, where the convolution approximately becomes a multiplication [6]:

$$\mathbf{Y}(\omega_k, \tau) = \mathbf{W}(\omega_k)\mathbf{X}(\omega_k, \tau), \quad k = 0, 1, \dots, K-1, \tag{9}$$

with K being the FFT length. The major benefit of this approach is the possibility to estimate the unmixing matrices $\mathbf{W}(\omega_k)$ for each frequency independently, however, at the price of possible permutation and scaling in each frequency bin. Solutions for those ambiguities can be found in [7, 8, 9, 10, 11] and [12, 13, 14, 15]. In the following we will concentrate on the possibilities for a fast parallel implementation using the CUDA framework. For comparison, we first discuss a reference Matlab implementation.

```
1  II = eye(observations);
2  for ii=1:frequency_bins
3    W = eye(observations);
4    X = spec(:,:,ii);
5    for n=1:iterations
6        Y = W * X;
7        delta_W = (II - sign(Y) * Y'/d_length) * W;
8        W = W + mu * delta_W;
9    end
10   WW(:,:,ii) = W;
11 end
```

Figure 2: Matlab code for the time-frequency domain ICA.

3. MATLAB IMPLEMENTATION

In Figure 2, a Matlab implementation of the time-frequency domain ICA is shown. It mainly consists of two `for`-loops which are used to iterate through all frequency bins and carry out the iterations of the gradient descent of equation (4). The Matlab syntax allows for an easy and convenient implementation of the gradient calculation. As shown in line seven, equation (5) and the calculation of the nonlinear score function from equation (6) translates to only one line of code.

In this code, the calculation of the expectation operator is carried out using a matrix multiplication `sign(Y) * Y'`. Using other programing languages this operation needs to be computed using a further `for`-loop. Therefore, using a single CPU, there are three major loops, with the innermost one hidden in the high-level data types. Actually, this loop needs the most of the computing power.

4. CUDA IMPLEMENTATION

4.1 CUDA Architecture

Before presenting the actual implementation, we first discuss the major differences between the CPU and GPU programing model. In Figure 3 a typical multi-core CPU model is shown. It consists of independent compute cores with own control logic and arithmetical-logical-unit (ALU). For speeding up the random memory access there exist a huge cache. On modern processors, this cache occupies up to half of the CPU's available transistors.

Figure 4 shows a typical GPU model. The major difference is the missing cache and a huge number of arithmetic units. In contrast to a CPU, several arithmetic units are controlled by just one control unit. This implicates, these groups of arithmetic units have to perform the same operations in every step.

Without any cache, the algorithms have to be carefully designed in order to access the memory in a regular and linear way. Even more, these access patterns have to be shared across the single arithmetic units, so memory coalescing is an important factor in every CUDA algorithm [21, 22]. In order to reduce the memory bandwith, all arithmetic units belonging to one control unit share a few thousand registers, which are dynamically allocated to the single threads by the runtime environment [22].

As discussed in the next section, there is a simple implementation of the frequency-domain ICA which meets all the requirements of the CUDA architecture.

4.2 Implementation

The CUDA framework allows only an implementation in C or C++. Not all C++ class specific constructions are allowed, as all class methods have to be inlined by the C++ compiler. In order to calculate the ICA, classes representing complex

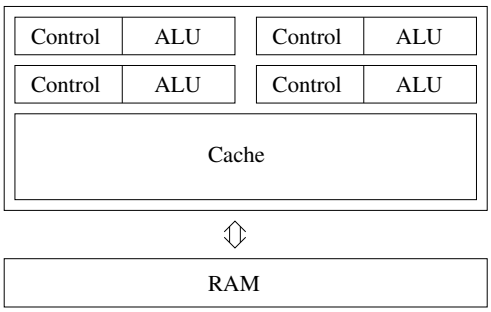

Figure 3: A typical 4-Core CPU model with independent cores and huge cache.

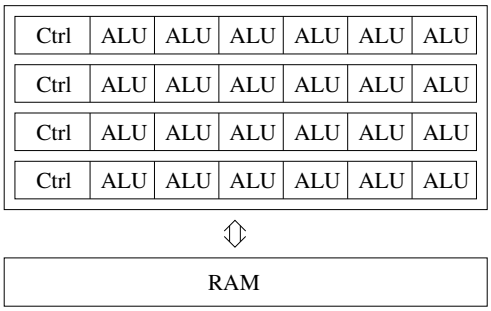

Figure 4: A typical multi-core GPU model with many coupled cores and no cache.

numbers (`cuComplex`), vectors (`cu2dVector`) and matrices (`cu2dMatrix`) have to be implemented. Besides all standard operations like addition, multiplication and absolute values, the `cu2dVector` needs a method for calculating a Kronecker product.

Using these classes the implementation of the kernel, which is executed independently on each CUDA arithmetic unit is very short, as shown in Figure 5.

Using this approach, a single kernel running on a single arithmetic unit executes the ICA for one frequency bin. Therefore the implementation lacks the outer `for`-loop, as these assignments are performed with help of the CUDA runtime environment. This is implemented in the first three lines, and follows the pattern from [22]. At line 6, the loop for the gradient descent is defined. The loop at the lines 8 to 11 is calculating the expectation value from equation (5). Finally, at line 13 the remaining operations for the gradient are executed. With C++ classes and overloaded operators, the formulation is almost as convenient as in Matlab.

As already mentioned, the calculation of the expectation operator is very time consuming. Fortunately, this calculation fits very good in the CUDA framework. In the case, of a (2×2)-system, each iteration in the lines 8-11 needs to access two complex values, which are stored consecutively in four float variables and can be transported from RAM in one operation. The calculation of the unmixed values, score function, and Kronecker product needs 36 multiplications, 32 additions, four divisions, and two square roots. With this high ratio of arithmetic operation to memory access, the single threads only seldom interfere while fetching next values and all arithmetic units are constantly occupied. Furthermore, the data for the individual frequency bins have the same size and can be easily interleaved. Therefore, a full coalesced access can be achieved [22].

When implementing this method as a mex-function for Matlab, data formats have to be converted, as Matlab's internal data organization is completely different. This transformation usually is fast and unproblematic. The whole source code with the mentioned Matlab connection is available at [23].

```
1  __global__ void par_ica_2d(icaData *d) {
2    int tid = threadIdx.x + blockIdx.x * blockDim.x;
3    if (tid < d->f_bins) {
4      cu2dMatrix W = (d->dev_w)[tid];
5      cuComplex  inv_dp(1.0f/(float)d->data_points);
6      for (int j=0; j < d->iterations; j++) {
7        cu2dMatrix tmp;
8        for (int i=0; i < d->data_points; i++) {
9          cu2dVect y = (W * d->data[tid + i*d->bins]);
10         tmp += (y.nonLinearF()).kProd(y.conj());
11       }
12     cu2dMatrix II(1,0,0,1);
13     W = (((II-(tmp * inv_mlen)) * W) * d->mu) + W;
14     }
15   d->dev_w[tid] = W;
16   }
17 }
```

Figure 5: Kernel code for the time-frequency domain ICA using CUDA framework.

5. SIMULATIONS

For the simulations different implementations of the ICA algorithms have been compared. The used data set from [24] consists of approximately 7.5s of speech. The tests have been performed using two and three speakers. The chosen parameters were a Hann window of length 2048, a window shift of 128, a FFT-length of 8192, and 200 iterations. Using these parameters, there were 4097 discrete frequency bins with 456 data points.

The used hardware was an Intel Q6600 with 2.4GHz and a Geforce 8800-GTS-512. Although this setup seems a little outdated, it is very good comparable to current available mobile hardware. Therefore, the results shown here, may be used as an estimate for real implementations.

In Table 1 the results of the simulations with two speakers are shown. At the first line the timings for the default Matlab implementation are displayed. The time needed for the separation was 123 seconds, which is approximately 17 times slower than real time. The next line shows a C++ implementation which is the same code as used for the CUDA implementation but executed on the CPU. Due to optimizations of the algorithm, it is approximately three times faster than the previous one. The measurements are divided into the setup time, which is needed for data conversion from the internal Matlab structures, and the actual ICA. The next two lines show the results when CUDA hardware is used. The difference in the implementations is the non-coalesced and coalesced memory access, which translates to a three times speed up. In the last case, the overall performance is 40 times better than the C++ and over 100 times faster than the Matlab implementation.

The CUDA implementation is about six times faster than real time in the (2×2)-case. Consequently, the remaining time may be used for solving the permutation and scaling problems and still achieve continuous processing.

The results for the (3×3)-case are quite similar, as shown in Table 2. Overall, there is an increase of computation time in the range of one and a half up to two times, which is due to the 50% more data to be processed.

	Setup	ICA	Total	RT-ratio
Matlab	-	123.60	123.6	16.93
C++	0.19	43.34	43.5	5.96
CUDA-NC	0.19	3.01	3.2	0.44
CUDA-C	0.19	0.91	**1.1**	**0.15**

Table 1: Run time in seconds for the different implementations for a dataset with two speech sources of approximate 7.5s length. RT-ratio shows the normalized computation time for a one second signal.

	Setup	ICA	Total	RT-ratio
Matlab	-	181.79	181.79	24.90
C++	0.28	82.94	83.22	11.40
CUDA-NC	0.28	6.43	6.71	0.92
CUDA-C	0.28	1.58	**1.86**	**0.25**

Table 2: Run time in seconds for the different implementations for a dataset with three speech sources of approximate 7.5s length.

6. CONCLUSIONS

In this paper we have proposed a CUDA implementation of the independent component analysis in the time-frequency domain. The ICA algorithm allows for an independent computation in each discrete frequency bin, and therefore can be easily adapted for the parallel CUDA architecture. The speedup, compared to a default CPU implementation, is in the order of two magnitudes. This new implementation allows for real time processing.

REFERENCES

[1] S.-I. Amari, A. Cichocki, and H. H. Yang. A new learning algorithm for blind signal separation. In *Advances in Neural Information Processing Systems*, volume 8, MIT Press, Cambridge, MA, 1996.

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

[3] J.-F. Cardoso and A. Soulomiac. Blind beamforming for non-Gaussian signals. *Proc. Inst. Elec. Eng., pt. F.*, 140(6):362–370, Dec. 1993.

[4] S. C. Douglas, H Sawada, and S. Makino. Natural gradient multichannel blind deconvolution and speech separation using causal FIR filters. *IEEE Trans. Speech and Audio Processing*, 13(1):92–104, Jan 2005.

[5] R. Aichner, H. Buchner, S. Araki, and S. Makino. Online time-domain blind source separation of nonstationary convolved signals. In *Proc. 4th Int. Symp. on Independent Component Analysis and Blind Signal Separation (ICA2003)*, pages 987–992, Nara, Japan, April 2003.

[6] P. Smaragdis. Blind separation of convolved mixtures in the frequency domain. *Neurocomputing*, 22(1-3):21–34, 1998.

[7] H. Sawada, R. Mukai, S. Araki, and S. Makino. A robust and precise method for solving the permutation problem of frequency-domain blind source separation. *IEEE Trans. Speech and Audio Processing*, 12(5):530–538, Sept. 2004.

[8] R. Mukai, H. Sawada, S. Araki, and S. Makino. Blind source separation of 3-d located many speech signals. In *2005 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 9–12, Oct 2005.

[9] R. Mazur and A. Mertins. An approach for solving the permutation problem of convolutive blind source separation based on statistical signal models. *IEEE Trans. Audio, Speech, and Language Processing*, 17(1):117–126, Jan. 2009.

[10] R. Mazur and A. Mertins. Simplified formulation of a depermutation criterion in convolutive blind source separation. In *Proc. European Signal Processing Conference*, pages 1467–1470, Glasgow, Scotland, Aug 2009.

[11] K. Rahbar and J. P. Reilly. A frequency domain method for blind source separation of convolutive audio mixtures. *IEEE Trans. Speech and Audio Processing*, 13(5):832–844, Sept. 2005.

[12] S. Ikeda and N. Murata. A method of blind separation based on temporal structure of signals. In *Proc. Int. Conf. on Neural Information Processing*, pages 737–742, 1998.

[13] K. Matsuoka. Minimal distortion principle for blind source separation. In *Proceedings of the 41st SICE Annual Conference*, volume 4, pages 2138–2143, 5-7 Aug. 2002.

[14] R. Mazur and A. Mertins. A method for filter shaping in convolutive blind source separation. In *Independent Component Analysis and Signal Separation (ICA2009)*, volume 5441 of *LNCS*, pages 282–289. Springer, 2009.

[15] R. Mazur and A. Mertins. Using the scaling ambiguity for filter shortening in convolutive blind source separation. In *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, pages 1709–1712, Taipei, Taiwan, April 2009.

[16] Francesco Nesta, Maurizio Omologo, and Piergiorgio Svaizer. A bss method for short utterances by a recursive solution to the permutation problem. In *Sensor Array and Multichannel Signal Processing Workshop, SAM 2008*, pages 357– 360, Darmstadt, Germany, July 2008.

[17] `http://www.nvidia.com/object/cuda_home_new.html`.

[18] S. Choi, A. Cichocki, and S. Amari. Flexible independent component analysis. In T. Constantinides, S. Y. Kung, M. Niranjan, and E. Wilson, editors, *Neural Networks for Signal Processing VIII*, pages 83–92, 1998.

[19] H. Brehm and W. Stammler. Description and generation of spherically invariant speech-model signals. *Signal Process.*, 12(2):119–141, 1987.

[20] I. Lee, T. Kim, and T.-W. Lee. Independent vector analysis for convolutive blind speech separation. In *Blind Speech Separation*, pages 169–192. Springer Netherlands, 2007.

[21] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1 edition, 7 2010.

[22] David B. Kirk and Wen mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach (Applications of GPU Computing Series)*. Morgan Kaufmann, 1 edition, 2 2010.

[23] `http://www.isip.uni-luebeck.de/index.php?id=479`.

[24] `http://www.kecl.ntt.co.jp/icl/signal/sawada/demo/bss2to4/index.html`.