Efficient Low-latency Convolution with Uniform Filter Partition and Its Evaluation on Real-time Blind Source Separation

Yui Kuriki, Taishi Nakashima, Kouei Yamaoka, Natsuki Ueno, Yukoh Wakabayashi, Nobutaka Ono*, Ryo Sato†
* Tokyo Metropolitan University, Tokyo, Japan
E-mail: kuriki-yui@ed.tmu.ac.jp
† RION CO., LTD., Tokyo, Japan

***Abstract*—In this paper, we discuss an efficient way to realize a low-latency convolution for real-time blind source separation (BSS). In some real-time applications, such as a hearing aid, both low complexity and low latency are necessary. To reduce the computation for the low-latency convolution,** *a partitioned convolution* **has been studied. It partitions a filter into multiple blocks, convolves each block with a signal via the frequency domain, and applies the overlap-add. In this paper, we focus on uniform partitioning as a suitable way and introduce it into real-time BSS. The complexity is estimated as the number of multiplications and evaluated with actual implementation on a Raspberry Pi 4B. The experimental results indicate that this approach can reduce the execution time of convolution calculation, and the optimum is obtained at a non-trivial block length.**

I. Introduction

In real-time blind source separation (BSS) [1]–[6], it is necessary to reduce both the computational complexity and the latency, especially in a small device such as a hearing aid. In many multichannel BSS, demixing matrices are estimated in the time-frequency domain, typically the short-time frequency transform (STFT) domain. Therefore, one way to reduce the latency is to use a short window. Although shortening the window leads to the degradation of the BSS performance, the combination with the dereverberation has been reported to show good performance [1]. Another approach is a two-path approach [2]. The demixing matrices are estimated in the time-frequency domain, while the source separation is conducted in the time domain by convoluting the input signal with quasi-causal finite impulse response (FIR) filters, which are obtained as the inverse fast Fourier transforms (FFTs) of the demixing matrices and truncation. However, the convolution in the time domain as the definition needs a considerable amount of computation.

Since convolution is a fundamental operation in digital signal processing, many studies have been performed for real-time implementation [7]–[14]. One of the basic ideas is to divide both the signal and the filter into small blocks, calculate their convolution via the frequency domain, and perform the over-lap add. This is referred to as *partitioned convolution*. However, to the authors' knowledge, its application to BSS has not been considered.

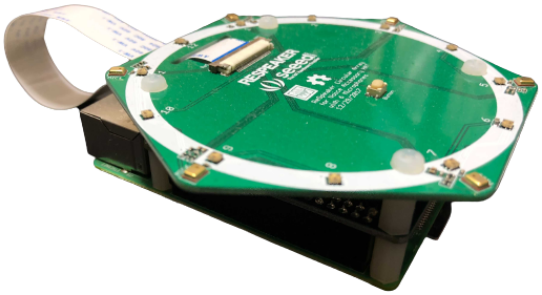

Fig. 1. A photograph of the Raspberry Pi 4B and microphone array.

In this study, to realize the low-latency online BSS in small devices [2], [15], we discuss how to reduce the computational complexity of the convolution by introducing the partitioned convolution. To avoid the complicated time-scheduling, we focus on a uniform partitioning and evaluate the computational complexity for a given acceptable latency. Also, we implement this method to the real-time online BSS on a Raspberry Pi 4B (Fig. 1) and evaluate the effectiveness by measuring processing time.

II. Methods for computing convolution

A. Convolution in time domain

Let $x[n]$, $h[n]$, and $y[n]$ be an input signal, an impulse response of an FIR filter with length L, and an output signal of the filter in the discrete time domain, respectively, where n denotes the discrete time.

The convolution operation is defined as

$$y[n] = \sum_{l=0}^{L-1} x[n-l]h[l]. \tag{1}$$

One way to compute the convolution is to calculate it as defined above. We refer to it as *linear convolution*.

Let us consider the latency of this calculation. We do not consider the intrinsic delay of the filter itself, but consider how many input samples are required to compute a new output sample, as the latency. Suppose that we already calculated $y[n-1]$ from $x[n-L+2], \ldots, x[n-1]$. Then, for newly

calculating $y[n]$, we only need to know $x[n]$. It means that we can calculate one more output sample if we obtain one more input sample. In this sense, this calculation does not bring any latency. This is good for low-latency processing and used in a low-latency real-time BSS system [2]. While, when we evaluate the computational complexity by the number of required multiplications, the computational complexity of the linear convolution is proportional to L. This amount of computation is not small, and constraints the length of the FIR filter for real-time processing.

B. Convolution via Frequency-domain with single block

As well known in digital signal processing, the convolution can be calculated via the frequency domain efficiently. Here we explain a well-used way to compute the convolution between an input signal with arbitrary length and an FIR filter with length L by using FFT [16]. We refer to this as *block convolution*.

First, we divide the input signal into segments of L samples (the same length of the filter) disjointly. Hereafter, we refer to the segment of L-sample input signal as a signal block. The ith signal block is defined as

$$\boldsymbol{x}_i = \begin{bmatrix} x[0+iL] & \cdots & x[L-1+iL] \end{bmatrix}. \tag{2}$$

In the same way, let us define a vector representation of the FIR filter as $\boldsymbol{h} = \begin{bmatrix} h[0] & \cdots & h[L-1] \end{bmatrix}$.

Then, L-point zeros are padded into both the signal block and the filter, and they are transformed into the frequency domain by the $2L$-point FFT. Let $\boldsymbol{X}_i$ and $\boldsymbol{H}$ be the discrete Fourier transform of the ith signal block and the filter, respectively. Using the Hadamard product between them as

$$\boldsymbol{Y}_i = \boldsymbol{X}_i \circ \boldsymbol{H}, \tag{3}$$

we obtain the convolution result of $\boldsymbol{x}_i$ and the filter $\boldsymbol{h}$ by the inverse FFT of $\boldsymbol{Y}_i$;

$$y_i[n] = (\boldsymbol{x}_i * \boldsymbol{h})[n]. \tag{4}$$

Finally, by the overlap-add with the result at the previous block, L-samples of the convolution result $y[n]$ between $x[n]$ and $h[n]$ are obtained as follows.

$$y[n+iL] = y_i[n] + y_{i-1}[n+L], n = 0, \ldots, L-1. \tag{5}$$

This method can be calculated with less complexity, thanks to the efficiency of FFT. On the other hand, this method requires a waiting time to collect the L-sample input signals, called "algorithmic latency".

III. Partitioned Convolution

A. Problem formulation

In this paper, we assume that $N < L$ samples are the acceptable latency of real-time convolution. We consider reducing the complexity of linear convolution between the N-sample input signal $x[n]$ and the L-sample filter $h[n]$. The conventional block convolution [16] (between the signal of any length and the L sample's filter) can implement with low latency by zero-padding, but the complexity increases. Therefore, we use the partitioned convolution [17]). In this method, the filter is also divided to reduce the complexity. Fig. 2 shows an overview of processing.

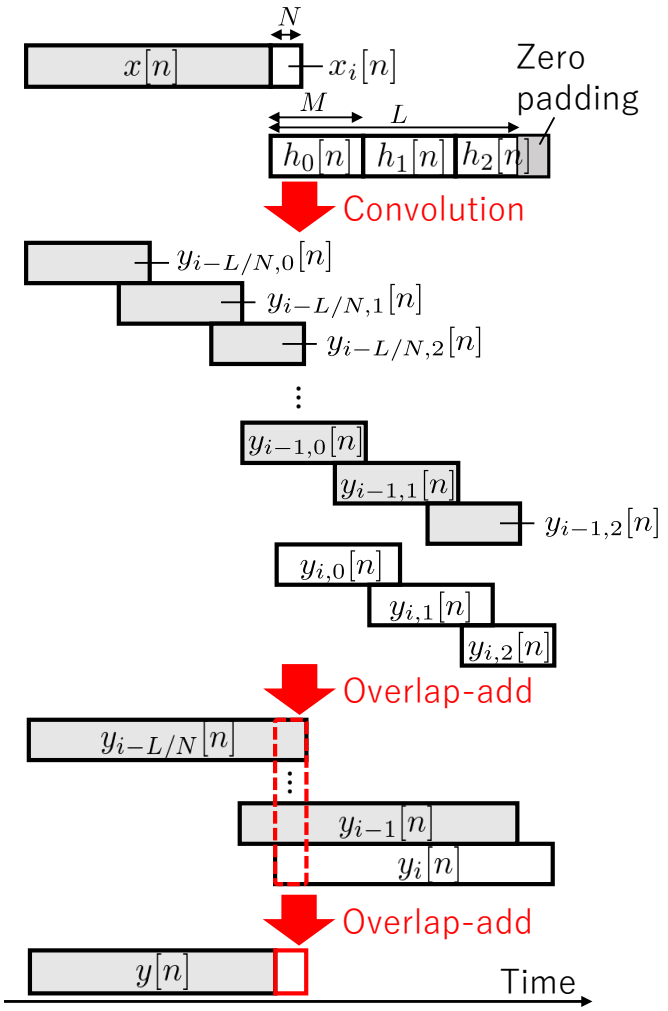

Fig. 2. Overview of the partitioned convolution. The input signal $\boldsymbol{x}$ and the FIR filter $\boldsymbol{h}$ are divided into multiple blocks $\boldsymbol{x}_i$ and $\boldsymbol{h}_j$, respectively. They are convolved via the frequency domain and overlap-added to obtain the output signal $y[n]$.

B. Algorithm

Here we consider dividing the input signal into multiple N-sample blocks where $N < L$ and re-define the signal block. The ith signal block is written as

$$\boldsymbol{x}_i = \begin{bmatrix} x[0+iN] & \cdots & x[N-1+iN] \end{bmatrix}. \tag{6}$$

In the partitioned convolution, the filter is also divided into several blocks. We divide the L-tap filter into multiple M-length blocks The jth filter block is written as

$$\boldsymbol{h}_j = \begin{bmatrix} h\,[0+jM] & \cdots & h[M-1+jM] \end{bmatrix}, \tag{7}$$

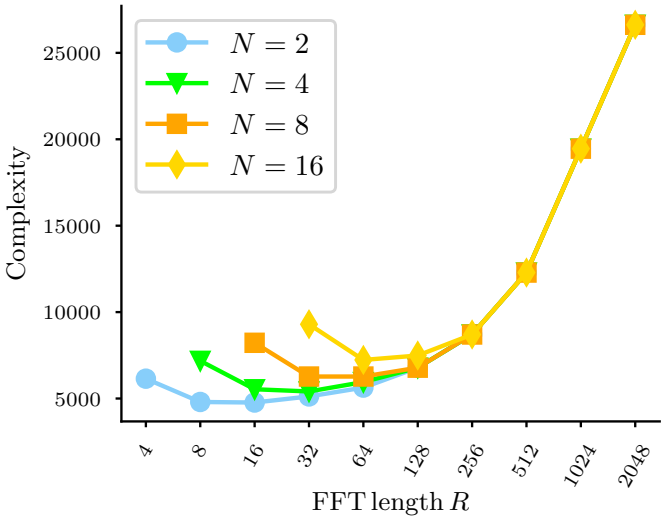

Fig. 3. The number of multiplications required for the partitioned convolution of an N-sample signal and a 1024-sample filter with various FFT lengths R.

where $h[n] = 0$ except for $n = 0, \ldots, L-1$. As a result, we have $J = \lceil L/M \rceil$ filter blocks, $\lceil * \rceil$ indicates the ceiling function.

First, calculate the $R = N+M$-point FFT of $\boldsymbol{x}_i$ and $\boldsymbol{h}_j$ that are zero-padded such that the length is R. Next, calculate the J convolutions of $\boldsymbol{x}_i$ and $\boldsymbol{h}_j (j = 0, \ldots, J-1)$ using Hadamard products and the inverse FFT. Then, calculate the convolution of $\boldsymbol{x}_i$ and the filter $h[n]$ by the overlap-add of J convolutions as the following equation:

$$y_i[n] = \sum_{j=0}^{J-1} (\boldsymbol{x}_i * \boldsymbol{h}_j)[n - jM], \tag{8}$$

where $(\boldsymbol{x}_i * \boldsymbol{h}_j)[n] = 0$ except for $n = 0, \ldots, N+M-1$, and $y_i[n] = 0$ except for $n = 0, \ldots, N+L-1$. Finally, further conduct the overlap-add of the current and previous convoluted signals as follows:

$$y[n] = \sum_{i=0}^{\infty} \sum_{j=0}^{J-1} (\boldsymbol{x}_i * \boldsymbol{h}_j)[n - iN - jM]. \tag{9}$$

By the partitioning and overlap-add of Eq. (9), we obtain the signal equivalent to the linear convolution result of input signal $x[n]$ and FIR filter $h[n]$.

The algorithmic latency of partitioned convolution is N samples. It equals the time for the input signal to accumulate for the signal block. Therefore, the shorter the signal block length, the lower the latency in processing. When $M = L$, and $J = 1$, this method becomes identical to the block convolution that does not divide the filter [16]. In this method, partitioning the filter with $M < L$ can further reduce the complexity.

C. Evaluation of complexity

In this paper, we assume that the FFT length R is the power of two, and the filter is time-invariant for simplicity. In the following, we do not consider the complexity required for the FFT of the filter since it can be calculated in advance.

Here, we consider the number of multiplications required to convolute one signal block in each process. First, the FFT of the signal block takes $(R/2)\log_2 R$ times of multiplication. Second, the Hadamard product between a signal block and J filter blocks takes $2JR$ times of multiplication. In calculating the Hadamard product, complex multiplication is carried out. It involves four multiplications at a time. So, $4R$ times of multiplication are required to calculate the Hadamard product between a signal block and each filter block. Considering that the negative frequency component is obtained by the complex conjugate of the positive frequency component, the total number of multiplications is $4R/2 = 2R$, which is done for the number of filter blocks J. Third, the inverse FFT takes $(JR/2)\log_2 R$ times of multiplication. It takes $(R/2)\log_2 R$ times in one block and performs J times. From the above, the multiplications of the entire partitioned convolution are as follows:

$$\frac{(1+J)R}{2}\log_2 R + 2JR. \tag{10}$$

On the other hand, the linear convolution for N samples takes NL times of multiplication. Thanks to the efficiency of FFT, the partitioned convolution can reduce the complexity compared to the linear convolution. Furthermore, we can see that Eq. (10) takes the minimum value at a specific R. As an example, the complexity for the different block size N and the the filter size $L = 1024$ is shown in Fig. 3. From this figure, it can be seen that the complexity is minimum when the FFT size $R = 4N$ for each latency N. Considering various implementation factors, R that minimizes Eq. (10) is not always optimal on the actual computer. But in any case, this equation suggests that further reducing complexity is possible by dividing the filter into blocks.

IV. Evaluation of Processing Time on Raspberry Pi

A. Experimental condition

In this experiment, we used a Raspberry Pi 4B, which was ran by the graphical user interface (GUI), and compared the calculation time for the convolution while changing the signal block length N and the FFT length R. We changed them as $N = 2^k$ for $k = 1, \ldots, 8$ and $R = 2^l$ for $l = k+1, \ldots, 11$, and correspondingly determined the filter block length as $M = R - N$. We implemented algorithms with `C++` and used the `FFTW` [18] for the FFT and the inverse FFT. The planner of the `FFTW` was set to `PATIENT`. The time measurement was done by `chrono`, the `C++` library and was evaluated a mean time of 10 trials for each condition. Note that the processing time includes the FFT of the signal, the calculation of the Hadamard product, the inverse FFT of the convolution result, and the overlap addition; unlike them, we did not include the calculation time of the FFT of the filter, assuming it is done in advance. The input signal was a speech with 10 s and the sampling frequency of 44.1 kHz. We used a low-pass filter with the cutoff frequency of 400 Hz and the tap length of 1024 samples as an FIR filter.

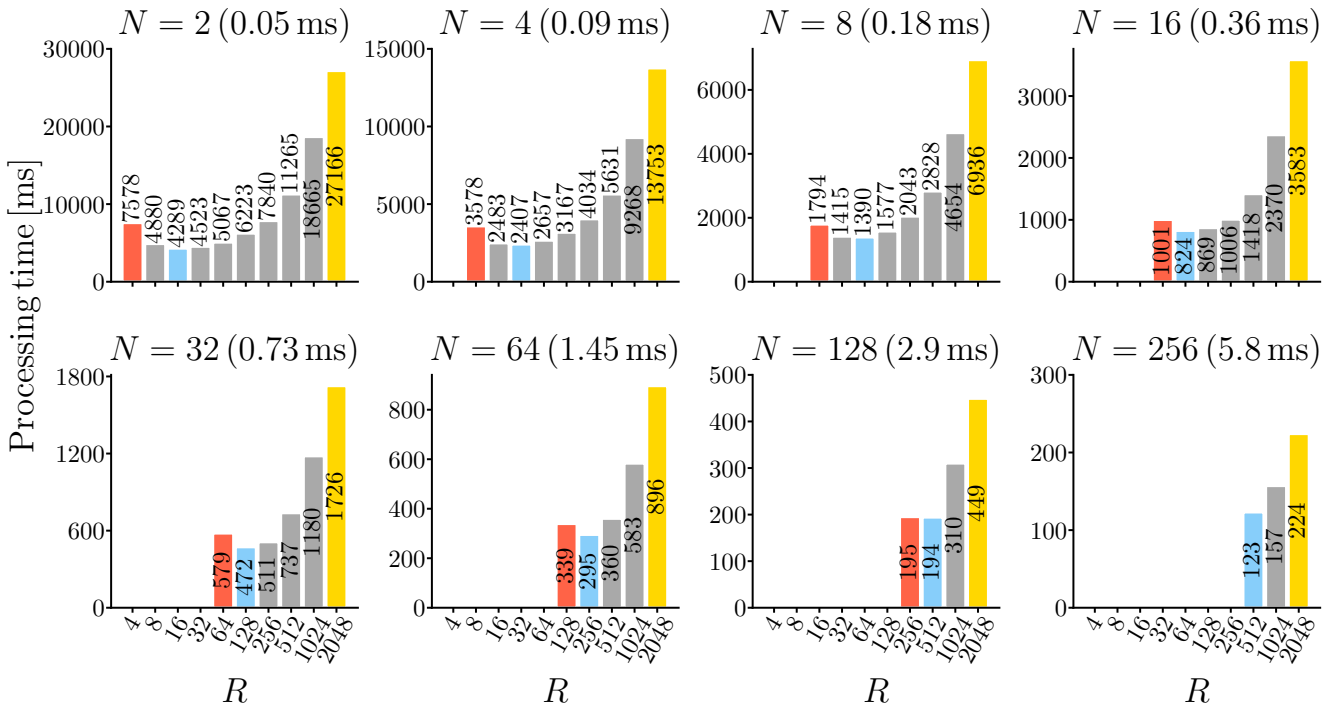

Fig. 4. Processing time of the partitioned convolution for different block size N and FFT length R. The orange bar (left end) shows the result when the filter block length is the same as the signal block length. The yellow bar (right end) shows the result when the filter is not partitioned. The blue bar represents the shortest processing time.

B. Results

Fig. 4 shows the processing time when the signal block length and the FFT length are changed. Comparing each signal block length N (each graph), the larger N is, the shorter the processing time. When the input signal and the FIR filter as described in Section IV-A were used, the processing time of the linear convolution in the time domain was 972 ms. Therefore, the processing time of the partitioned convolution is shorter than that of the linear convolution when $N = 16$ or longer with an adequate FFT length R. Then, focusing on the FFT length (horizontal axis), each signal block length has an FFT length which is the shortest processing time (blue-colored bar). When $N \leq 8$, the optimum FFT length was $R = 8L$, and when $16 \leq N \leq 128$, the optimum FFT length was $R = 4L$. Since the filter block length does not affect the algorithmic latency, we can choose the filter block length $M = R - N$ that will be the shortest processing time at each N.

A comparison of Fig. 3 and Fig. 4 shows that the processing time is similar to the complexity. Note that this result may change depending on the implementation.

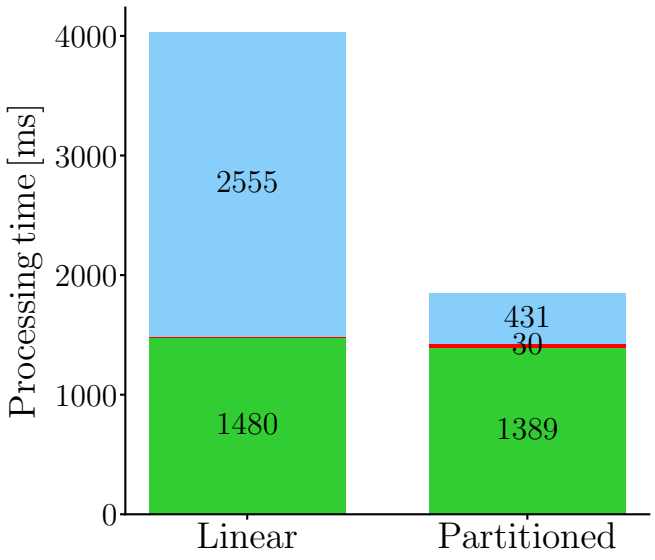

Fig. 5. Processing time of real-time BSS by the linear convolution in the time domain (left) and the partitioned convolution with the optimum block size (right). The breakdowns of the bars represent three processes: the AuxIVA (green), the convolution (blue), and the FFT of the separation filter (red).

V. Evaluation of Usefulness in Real-time BSS

A. Experimental condition

In this experiment, we applied the partitioned convolution to the low-latency real-time BSS implemented on the Raspberry Pi 4B and measured the processing time. In this simulation, we used the character-based user interface (CUI) for staring the system on Raspberry Pi 4B to ignore the processing time for unnecessary operations e.g., drawing graphics. The programming language and libraries were the same as the experiment described in Section IV. Our real-time BSS is based on online auxiliary-function-based independent vector analysis (AuxIVA) [2]. The filter for separation is estimated

in frequency domain, transformed into time domain by FFT, and truncated the non-causal components for realizing the low latency. In the original system [2], the resultant FIR filters for separation are convolved with the input signal in time domain by linear convolution. We replaced this by the partitioned convolution, and compared the processing time.

We set the filter length as $L = 512$ samples and updated it every time when the 1024-sample input signal was stored in the buffer. The number of microphones was four and the sampling frequency was $16\,\mathrm{kHz}$. The input signal to the system was a mixture of two speeches uttered by different speakers, and its length was $20\,\mathrm{s}$. We set the allowable delay as 64 samples; that is, the signal block size N was 64 samples. By minimizing Eq. (10) for given L and N, we chose $R = 256$ samples as the optimum FFT length for redusing the complexity. After that, we determined the the filter block size as $M = R - N = 192$. Although the AuxIVA calculates the separation filter for four outputs, this implementation outputs one separated source. Therefore, the processing time of filtering for only one output was measured. We evaluated an average of the processing time over 10 trials for each method.

B. Results

Fig. 5 shows that the processing time spent per the AuxIVA filter update (green), the convolution (blue), and the FFT of the separation filter (red). The processing time of the FFT of the filter is present only in the partitioned convolution. This is because the partitioned convolution requires the re-transform of a time-domain quasi-causal filter by [2] to the frequency-domain one, unlike the linear convolution. Both have processing times shorter than $20\,\mathrm{s}$, allowing for real-time processing. The AuxIVA takes about $1400\,\mathrm{ms}$. In the linear convolution, convolution calculation accounts for $63\,\%$ of the total processing time. On the other hand, the partitioned convolution reduced the processing time of the convolution (including the FFT of the filter) to about $18\,\%$. This is about $25\,\%$ of the total processing time. Further processing time savings can be expected if the FFTs of partitioned filters are directly estimated in AuxIVA. It will be investigated in future.

Even though the linear convolution in the time domain theoretically yields no algorithm delay, the actual system needs some waiting time for other reasons. For example, the audio signal is usually stored in an audio buffer and then processed. In our system, the buffer size is typically about 64 samples. It indicates that, in the case of our system, the partitioned convolution with up to 64-sample filter block does not increase the actual delay compared to the linear convolution and only reduces the computational complexity. This is a strong advantage of introducing the partitioned convolution to the real-time BSS.

VI. Conclusions

In this paper, we introduced the partitioned convolution for efficiently realizing low-latency real-time blind source separation. Adding to the evaluation of computational complexity by the number of multiplication, we implemented the partitioned convolution on the Raspberry Pi and confirmed the reduction in the processing time. Reducing the amount of computation will allow us to allocate device resources for other purposes, such as increasing the number of channels, using longer filters, and presenting the separated signal in stereo to maintain a sense of localization.

Acknowledgment

This work was supported by JST CREST Grant Number JPMJCR19A3, Japan.

References

[1] T. Ueda, T. Nakatani, R. Ikeshita, K. Kinoshita, S. Araki, and S. Makino, "Low latency online blind source separation based on joint optimization with blind dereverberation," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 506–510.

[2] M. Sunohara, C. Haruta, and N. Ono, "Low-latency real-time blind source separation for hearing aids based on time-domain implementation of online independent vector analysis with truncation of non-causal components," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 216–220.

[3] T. Taniguchi, N. Ono, A. Kawamura, and S. Sagayama, "An auxiliary-function approach to online independent vector analysis for real-time blind source separation," in *Proceedings of Joint Workshop on Hands-free Speech Communication and Microphone Arrays (HSCMA)*, 2014, pp. 107–111.

[4] J. Chua, G. Wang, and W. B. Kleijn, "Convolutive blind source separation with low latency," in *Proceedings of IEEE International Workshop on Acoustic Signal Enhancement (IWAENC)*, 2016, pp. 1–5.

[5] J. Chua and W. B. Kleijn, "A low latency approach for blind source separation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 8, pp. 1280–1294, 2019.

[6] P. Magron and T. Virtanen, "Online spectrogram inversion for low-latency audio source separation," *IEEE Signal Processing Letters*, vol. 27, pp. 306–310, 2020.

[7] A. Torger and A. Farina, "Real-time partitioned convolution for ambiophonics surround sound," in *Proceedings of IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2001, pp. 195–198.

[8] W. G. Gardner, "Efficient convolution without input/output delay," *journal of the audio engineering society*, 1994.

[9] J.-S. Soo and K.K. Pang, "Multidelay block frequency domain adaptive filter," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 2, pp. 373–376, 1990.

[10] C. Burrus, "Block realization of digital filters," *IEEE Transactions on Audio and Electroacoustics*, vol. 20, no. 4, pp. 230–235, 1972.

[11] N. Jillings, J. D. Reiss, and R. Stables, "Zero-delay large signal convolution using multiple processor architectures," in *Proceedings of IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2017, pp. 339–343.

[12] G. P. M. Egelmeers and P. C. W. Sommen, "A new method for efficient convolution in frequency domain by nonuniform partitioning for adaptive filtering," *IEEE Transactions on Signal Processing*, vol. 44, no. 12, pp. 3123–3129, 1996.

[13] G. Garcia, "Optimal filter partition for efficient convolution with short input/output delay," *journal of the audio engineering society*, 2002.

[14] S. Siddiq, "Optimization of convolution reverberation," 2020, pp. 46–53.

[15] C. Haruta, N. Ono, and Y. Kinoshita, "Framewise finite impulse response filtering based on time-frequency mask for low-latency speech enhancement," in *Proceedings of Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2021, pp. 1215–1220.

[16] T. G. Stockham, "High-speed convoution and correlation," in *Proceedings of the April 26-28, 1966, Spring Joint Computer Conference*, 1966.

[17] F. Wefers and M. Vorländer, "Optimal filter partitions for real-time fir filtering using uniformly-partitioned fft-based convolution in the frequency-domain," in *Proceedings of the 14th International Conference on Digital Audio Effects (DAFx-11)*, 2011, pp. 155–161.

[18] M. Frigo and S. G. Johnson, "FFTW," https://www.fftw.org/, (Accessed on 01/27/2022).