INTERSPEECH 2023
20-24 August 2023, Dublin, Ireland

GPU-accelerated Guided Source Separation for Meeting Transcription

Desh Raj[1], *Daniel Povey*[2], *Sanjeev Khudanpur*[1,3]

[1]CLSP & [3]HLTCOE, Johns Hopkins University, Baltimore, USA; [2]Xiaomi Corp., Beijing, China

draj@cs.jhu.edu, dpovey@gmail.com, khudanpur@jhu.edu

Abstract

Guided source separation (GSS) is a target-speaker extraction method that uses pre-computed speaker activities and blind source separation to perform front-end enhancement of overlapped speech signals. First proposed during the CHiME-5 challenge, it provided significant improvements over the delay-and-sum beamforming baseline. Despite its strengths, the method has seen limited adoption for meeting transcription benchmarks primarily due to its high computation time. In this paper, we describe our improved implementation of GSS that leverages the power of modern GPU-based pipelines, such as batched processing of frequencies and segments, to provide 300x speed-up over CPU-based inference. This allows us to perform detailed ablation studies over several parameters of the GSS algorithm — context duration, number of channels, and noise class, to name a few. We provide reproducible pipelines for speaker-attributed transcription of popular meeting benchmarks: LibriCSS, AMI, and AliMeeting. Our code is publicly available at: `https://github.com/desh2608/gss`.

Index Terms: multi-talker ASR, GSS, speaker diarization.

1. Introduction

Automatic speech recognition (ASR) for meetings is characterized by overlapping speech and far-field multi-channel audio [1]. Speaker overlaps, in particular, result in severe degradation in transcription accuracy, both as a result of inaccurate detection of overlapping segments [2, 3], as well as increased ASR errors on these segments [4, 5, 6]. With the rise of deep neural networks (NNs), there have been several advancements in using NN-based mask estimation methods for speech separation [7, 8]. However, these methods are often limited to fully overlapped synthetic speech, and fail to generalize to real, sparse overlaps that are common in multi-talker meetings [9, 10]. Recently, an alternate formulation of speech separation methods, named continuous speech separation (CSS), targeted specifically for sparse overlaps containing an unknown number of speakers, has been proposed [11, 12].

Despite growing popularity of supervised methods, beamforming of multi-channel signals using unsupervised mask estimation remains a strong baseline for multi-talker ASR [13, 14, 15, 16]. Among these, the recently proposed guided source separation (GSS) stands out as a particularly effective approach for handling noisy, overlapping speech using diarization information [17, 18]. The method was first proposed for the CHiME-5 challenge, where it provided relative word error rate (WER) improvement of 21.1% on the multi-array track using oracle segmentation [17]. It was later adopted as the challenge baseline for CHiME-6, and used by the winning systems on both oracle and unsegmented tracks [19, 20, 21, 22].

GSS relies on fundamental ideas from blind source separation (BSS), using spatial mixture models to model the sum of short-time Fourier transform (STFT) bins of multiple speakers [23]. It uses diarization information to (i) estimate the number of mixture components, and (ii) avoid the speaker-frequency permutation problem when processing different frequency bins independently. We will describe the algorithm in detail in Section 2. However, despite its strong performance in the CHiME-5 and CHiME-6 challenges, GSS has seen limited adoption in other multi-talker benchmarks, most notably offline meeting transcription, primarily due to its significant computational cost. For instance, enhancing the CHiME-6 `dev` set using 80 CPU jobs requires approximately 20 hours with the original GSS implementation. There have been some efforts to adapt the offline GSS algorithm for real-time enhancement by relying on limited right context [24], but these are also CPU-bound.

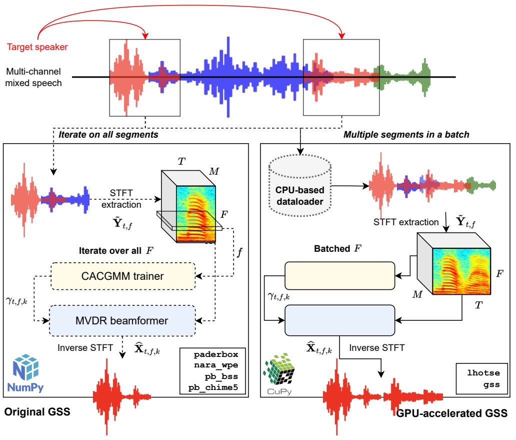

Figure 1: *Overview of batch processing for GPU-accelerated GSS. Solid and dotted lines denote GPU-bound and CPU-bound operations, respectively. The WPE module is not shown.*

In this paper, we describe our new, publicly-available GPU-accelerated implementation of GSS that aims to remove this computational bottleneck of enhancement. We achieve this primarily by porting all the computations on the GPU, and applying batching at several levels to maximize the GPU memory utilization. Our implementation is inspired by modern deep learning pipelines where background CPU-based workers perform data loading of large tensors, while the data processing is performed by GPUs [25]. The resulting 300x speedup allows us to perform ablation experiments using several benchmarks to analyze the importance of factors such as WPE, noise class, context duration, number of BSS iterations, and number of channels, towards GSS performance.

Finally, we provide complete reproducible recipes for meeting transcription of several benchmarks, namely LibriCSS, AMI, and AliMeeting. This includes diarization with and without overlap assignment, GSS-based enhancement, and pretrained models for ASR inference with neural transducers. We believe that our results will provide strong reproducible baselines for all future work on speaker-attributed ASR.

2. Guided source separation

We first provide an overview of the GSS algorithm, as proposed in [17]. Consider a multi-channel input recording provided in the

10.21437/Interspeech.2023-42

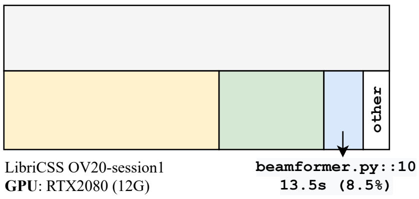

Figure 2: *Representative output of profiler during enhancement of a single recording.*

form of STFT features $\mathbf{Y}_{t,f} \in \mathbb{C}^M$, where t and f are time and frequency bins, respectively, and M is the number of channels. The GSS algorithm assumes the following model of the signal:

$$\mathbf{Y}_{t,f} = \underbrace{\sum_{k\in K}\mathbf{X}^{\text{early}}_{t,f,k}}_{\mathbf{X}^{\text{early}}_{t,f}} + \underbrace{\sum_{k\in K}\mathbf{X}^{\text{tail}}_{t,f,k}}_{\mathbf{X}^{\text{tail}}_{t,f}} + \mathbf{N}_{t,f}, \tag{1}$$

where K is the number of speakers in the recording, and "early" and "tail" refer to components of the reverberation. $\mathbf{N}_{t,f}$ is the STFT component due to noise. For target-speaker extraction, the objective is to estimate the de-reverberated signal from a desired speaker k, i.e., $\widehat{\mathbf{X}}_{t,f,k}$. This estimation is performed in three steps, as described below.

1. **De-reverberation using WPE.** First, we estimate $\mathbf{X}^{\text{tail}}_{t,f}$, i.e., the "tail" part of the reverb, using the popular weighted prediction error (WPE) algorithm [26, 27], and remove it from the signal, followed by normalization to get unit STFT vectors, i.e.,

$$\tilde{\mathbf{Y}}_{t,f} = \frac{\mathbf{Y}_{t,f} - \widehat{\mathbf{X}}^{\text{tail}}_{t,f}}{\|\mathbf{Y}_{t,f} - \widehat{\mathbf{X}}^{\text{tail}}_{t,f}\|}. \tag{2}$$

2. **Mask estimation using CACGMMs.** In the second stage, STFT masks are estimated for each speaker (and noise). The mask estimation technique is based on the "sparsity assumption," which assumes that only one speaker is active in each time-frequency bin. Using this assumption, the vector in each T-F bin can be assumed to have been generated from a mixture model where each component of the mixture belongs to a different speaker (or noise class). In the case of GSS, each mixture component is a complex angular central Gaussian (CACG), and hence the mixture model is a CACGMM [28]. A CACGMM models sums of unit-normalized complex-valued random variables, and the probability density function at a frequency index f is determined as

$$p(\tilde{\mathbf{Y}}_{t,f}) = \sum_{k\in K}\pi_{f,k}\mathcal{A}(\tilde{\mathbf{Y}}_{t,f};\mathbf{B}_{f,k}), \tag{3}$$

where $\pi_{f,k}$ is the mixture weight of source k at frequency index f, and $\mathcal{A}(\mathbf{y};\mathbf{B})$ is a CACG distribution parameterized by $\mathbf{B} \in \mathbb{C}^{M\times M}$:

$$\mathcal{A}(\mathbf{y};\mathbf{B}) = \left(\frac{1}{2\pi}\right)^M \frac{(M-1)!}{|\mathbf{B}|}(\mathbf{y}^H\mathbf{B}^{-1}\mathbf{y})^{-M}, \tag{4}$$

where $(\cdot)^H$ denotes the Hermitian transpose. Mixture model parameters are usually estimated using the EM algorithm that alternates between estimating the state posteriors (in the E-step) and the parameters of the component model (in the M-step). However, there are two problems in applying EM independently for each frequency bin: (i) the number of sources K is unknown; and (ii) the same mixture component may correspond to different sources in different frequency bins. GSS solves both of these problems by assuming that speaker activities are known for the recording, either through an oracle or a diarization system. Given the speaker activities $a_{t,k} \in \{0,1\}$, we convert the time-invariant mixture weights to time-varying weights as

$$\pi_{t,f,k} = \frac{\pi_{f,k}a_{t,k}}{\sum_{k'\in K}\pi_{f,k'}a_{t,k'}} \tag{5}$$

There may still be a permutation problem between the mixture components for the target speaker and the noise signal, since noise is present throughout the recording. To solve this problem, the GSS algorithm adds a "context window" to each utterance. We run the EM algorithm on the CACGMM until convergence to obtain the final state posteriors $\gamma_{t,f,k}$ as the estimated speaker masks.

3. **Mask-based MVDR beamforming.** Finally, we compute the spatial covariance matrices for the target and background signals as

$$\Phi_k(f) = \frac{1}{T}\sum_t \gamma_{t,f,k}\tilde{\mathbf{Y}}_{t,f}\tilde{\mathbf{Y}}^H_{t,f}, \tag{6}$$

$$\Phi_{\text{bg}}(f) = \frac{1}{T}\sum_t\left(\sum_{k'\neq k}\gamma_{t,f,k'}\right)\tilde{\mathbf{Y}}_{t,f}\tilde{\mathbf{Y}}^H_{t,f}, \tag{7}$$

which are then used to compute the minimum-variance distortionless response (MVDR) filter [29, 30] as

$$\mathbf{h}_k(f) = \frac{\Phi^{-1}_{\text{bg}}(f)\Phi_k(f)\mathbf{e}_{\text{ref}}}{\text{tr}\left(\Phi^{-1}_{\text{bg}}(f)\Phi_k(f)\right)}, \tag{8}$$

where $\mathbf{e}_{\text{ref}} \in \{0,1\}^M$ is a one-hot vector indicating the reference channel, selected to maximize the signal-to-noise ratio. Finally, the enhanced STFT signal is computed as $\widehat{\mathbf{X}}_{t,f,k} = \mathbf{h}_k(f)^H\tilde{\mathbf{Y}}_{t,f}$.

3. GPU-accelerated inference

The original GSS implementation[1] is slowed down by the following key factors: **(A)** All computations (i.e., feature extraction, WPE, mask estimation, beamforming, and iSTFT) are performed on the CPU using NumPy [31]. **(B)** For each segment, the CACGMM-based mask estimation is performed by iterating over all frequency bins (usually 513) sequentially. **(C)** A context window (usually 15s) is used for all segments regardless of the segment duration, resulting in a lot of wasted computation for short segments. **(D)** All the segments are processed sequentially, so processing time for a recording increases linearly with number of identified segments. A workaround for limitation **(D)** was provided by using MPI-based multi-processing (or Kaldi-style parallelization[2]) to enhance segments concurrently on a multi-node CPU cluster. Nevertheless, enhancing the CHiME-6 `dev` set, for instance, may require close to 20 hours (wall clock time) even using 80 CPU jobs (§ 5.4).

We propose to accelerate GSS-based inference by leveraging the power of modern GPU hardware and pipelines inspired from neural network training. First, to address limitation **(A)**, we use CuPy arrays which speed up array operations significantly using CUDA kernels, compared with regular NumPy-based array operations [32]. Since the most computationally intensive operations in the pipeline (such as CACG probability estimation) involve matrix multiplications (through `einsum`), GPU-based CUDA kernels are more efficient. However, simply transferring all arrays to CuPy is not sufficient — for example, limitations **(B)**–**(D)** still require sequential processing, which limits GPU utilization. To maximize GPU utilization and improve real-time factor (RTF), we perform the following additional optimizations.

1. **Segment batching.** Instead of processing each segment independently, we batch together multiple segments for inference. However, unlike neural network based training pipelines where batching is performed by stacking sequences in parallel, our batches are formed by concatenating segments sequentially along the time (T) axis to create "super-segments." We choose this form of batching because (i) the `einsum`-based operations are designed to work with 3-D tensors, and (ii) parallel batching of segments with padding would result in wasted memory. Since

[1] https://github.com/fgnt/pb_chime5

[2] https://kaldi-asr.org/doc/queue.html

multiple components of the inference (such as mask estimation and beamforming) compute statistics over the entire segment, we always create super-segments of the same recording with the same target speaker. Furthermore, we only use a single context window for the entire batch (instead of segment-wise context), which further reduces the wasted computations for short segments. This batching technique should work well for the case when optimal reference channels do not vary over the duration of the recording (i.e., when speakers are stationary, which is common for meeting scenarios)[3].

2. **CPU-based data-loaders.** We ensure that GPU idle time is minimized by off-loading the batch creation process to CPU-based data-loaders (possibly containing multiple workers), similar to deep learning pipelines.

3. **Frequency batching.** To address limitation **(B)**, we modified the CACGMM-based mask estimation to process 3-D tensors (F,T,M) instead of 2-D arrays (T,M). This simple change allows us to process all the frequency bins concurrently in a batch, significantly increasing GPU memory utilization.

4. **Einsum path optimization.** As mentioned above, several components in the GSS pipeline are implemented using `einsum`, and uses an optimal path contraction technique to find the path of minimum floating-point operations through the sequence (often resulting in up to 15x speed-up over a naive computation) [33]. However, the optimal path finding itself is computationally demanding, with a complexity of $\mathcal{O}(N!)$ for N arrays, and since it is performed several times during inference (for example, in each iteration of the CACGMM inference), it overshadows any speed-ups from the actual contracted sum. To remedy this, we cache the optimal computed path in the first iteration and re-use it in subsequent iterations.[4]

Once the enhanced waveform is obtained for the super-segment, we use background worker threads to chunk it into the original segments and save the audios to disk. With all these speed-ups, we were able to enhance a 10-minute LibriCSS recording in 159s (as shown in Fig. 2). This is equivalent to a real-time factor (RTF) of approximately 0.3. Further speed-ups can be obtained using GPUs with larger memory, by using bigger batches. From an implementation perspective, we can divide the pipeline into two parts. The *data processing* part is tasked with efficiently creating segments and corresponding speaker activities, while the *inference* part performs the actual computations on GPU. We use Lhotse for all data processing, i.e., to store and read recording metadata, to represent speaker activities, and to perform segment batching to create super-segments [34]. Since we use Lhotse's supervision manifests to store speaker activities, it allows us to use either oracle segments, or read segments from RTTM files (diarization output) with the same data processing pipeline (cf. § 5.1 and 5.2).

4. Experimental Setup

4.1. Data

We performed evaluations on three publicly-available meeting datasets: LibriCSS, AMI, and AliMeeting. **LibriCSS** consists of multi-channel audio recordings of 8-speaker "simulated conversations" that were created by combining utterances from the LibriSpeech `test-clean` set [35]. It comprises 10 one-hour long sessions, each of which is made up of six 10-minute "mini sessions" that have different overlap ratios (ranging from 0% to 40%). **AMI** (Augmented Multi-party Interactions) consists of 100 hours of recorded meetings containing 4 or 5 speakers per session [36].

Table 1: *Statistics of datasets used for evaluations. The k-speaker durations are in terms of fraction of total speaking time.*

	LibriCSS		**AMI**			**AliMeeting**		
	Dev	**Test**	**Train**	**Dev**	**Test**	**Train**	**Eval**	**Test**
Duration (h:m)	1:00	9:05	79:23	9:40	9:03	111:21	4:12	10:46
Num. sessions	6	54	133	18	16	209	8	20
Silence (%)	6.2	6.7	18.1	21.5	19.6	7.11	7.7	8.0
1-speaker (%)	81.3	81.2	75.5	74.3	73.0	52.5	62.1	63.4
2-speaker (%)	18.6	18.5	21.1	22.2	21.0	32.8	27.6	24.9
>2-speaker (%)	0.1	0.4	3.4	3.5	6.0	14.7	10.2	11.7

AliMeeting is a Mandarin-language corpus collected from real meetings, originally designed for ICASSP 2022 M2MeT challenge [37]. Each session consists of a 15 to 30-minute discussion by 2-4 participants. Detailed statistics for all datasets are shown in Table 1.

We used three different mic settings for our experiments: IHM (individual headset microphone), SDM (single distant microphone), and GSS (GSS-enhanced multi-mic). Since LibriCSS does not provide headset recordings, we used the corresponding LibriSpeech utterances concatenated together to simulate IHM. For all datasets, the first channel of the first array was used for the SDM setting. For LibriCSS and AliMeeting, we used all available channels for GSS, whereas for AMI, we used the first of the two arrays.

4.2. Models

We trained separate transducer-based ASR models for each benchmark. For LibriCSS, we used a pretrained Conformer-transducer [38] trained on LibriSpeech. For AMI and AliMeeting, we trained a Zipformer [39] transducer on a combination of IHM, IHM with simulated reverb, SDM, and GSS-enhanced far-field recordings of the corresponding train set, and the resulting model was used to evaluate all microphone settings. In all cases, we applied three-fold speed perturbation and noise augmentation using MUSAN [40] noises. We used a "stateless" decoder consisting of a convolutional layer with a bi-gram context. The model was trained using a pruned RNN-T loss [41] implemented in k2[5]. For decoding, we used a WFST-based parallel beam search method [42].

For the non-oracle segmentation experiments in § 5.2, we used a multi-class spectral clustering based diarization system with and without overlap assignment [43, 44]. The system consists of a Pyannote-based speech activity detector [45] fine-tuned on the corresponding train set for AMI and AliMeeting. For embedding extraction, we used a pretrained ResNet101-based x-vector model [46]. For these experiments, we report diarization error rates (DER) and concatenated minimum-permutation WER (cpWER) [19] in order to analyze the impact of diarization errors on downstream ASR. We did not use any collars to compute DERs for LibriCSS and AMI, but a collar of 0.25 was used for AliMeeting following the original work. All diarization recipes, generated RTTM files, and inference pipelines for meeting transcription are publicly available[6].

5. Results & Discussion

5.1. Far-field ASR

We first demonstrate the improvement in far-field ASR performance when using GSS with oracle segmentation, as shown in Table 2. The IHM and SDM settings may be considered as the lower and upper bounds on WER (or CER), respectively. We found that across all the datasets, GSS improved ASR performance significantly, with the **recovered error rates**[7] **being 86.8%, 65.9%, and 80.4%** for LibriCSS, AMI, and AliMeeting, respectively. As expected, most

[3] We also provide the option for using at most one segment per batch, for the case when speakers are not stationary (§ 5.4).

[4] In practice, since our tensor dimensions often have the same relative order across all batches (i.e., $M<F<T$), we can simply fix the optimal path for all `einsum` operations.

[5] `https://github.com/k2-fsa/k2`

[6] `https://github.com/desh2608/icefall/tree/multi_talker`

[7] $(W_{\text{SDM}} - W_{\text{GSS}})/(W_{\text{SDM}} - W_{\text{IHM}})$

Table 2: *Comparison of close-talk and far-field ASR performance. The GSS setting uses 7 channels for LibriCSS and 8 channels for AMI and AliMeeting. [†]LibriCSS IHM refers to the corresponding LibriSpeech utterances. [#]For AliMeeting, the numbers are CER.*

Dataset	Setting	Ins.	Del.	Sub.	WER
LibriCSS	IHM[†]	0.3	0.2	1.7	2.2
	SDM	1.1	3.1	6.6	10.8
	GSS	0.3	0.9	2.1	3.3
AMI	IHM	2.2	4.5	11.3	18.0
	SDM	4.0	9.6	18.5	32.1
	GSS	2.4	6.1	14.3	22.8
AliMeeting[#]	IHM	1.0	3.8	7.3	12.1
	SDM	2.0	10.0	14.4	26.4
	GSS	1.1	4.9	9.0	15.0

Table 3: *Effect of GSS-based enhancement on unsegmented speaker-attributed ASR performance, measured by cpWER (%). ✗ and ✓ correspond to the SDM and GSS settings from Table 2, respectively.*

	Diarizer	DER				GSS	cpWER			
		FA	MS	Conf.	Total		Ins.	Del.	Sub.	Total
LibriCSS	Spectral	1.2	10.4	3.4	14.9	✗	1.0	13.6	3.7	18.3
						✓	0.7	12.3	2.8	15.9
	+ OVL	2.2	3.8	5.3	11.3	✗	2.6	8.1	6.4	17.1
						✓	1.6	7.1	3.4	**12.1**
AMI	Spectral	3.2	18.2	4.1	25.5	✗	2.6	20.3	15.5	38.5
						✓	2.6	18.0	13.0	33.6
	+ OVL	7.4	9.6	6.7	23.7	✗	4.4	14.5	19.7	38.5
						✓	3.6	12.2	15.2	**31.0**
AliMeeting	Spectral	0.2	13.6	2.6	16.4	✗	1.2	26.4	10.1	37.6
						✓	0.9	24.3	7.2	32.4
	+ OVL	2.8	6.0	5.6	14.4	✗	2.3	18.8	14.3	35.4
						✓	1.7	17.0	9.8	**28.5**

of the improvement was obtained from recovered deletion and substitution errors, possibly from better recogntion of overlapped speech segments.

5.2. Effect of diarization

For meeting transcription, it may be hard to obtain oracle segmentation, and often a diarization system is used as a pre-processing step for ASR. In Table 3, we investigate the impact of using non-oracle segmentation with GSS-based enhancement. We found that **when no enhancement is performed, overlap detection results in little to no cpWER improvement**, since the ASR system is unable to handle overlapping segments. This finding corroborates the results of the winning CHiME-6 system [22], which was able to substantially improve ASR performance on unsegmented recordings using TS-VAD based diarization [47]. Using GSS results in significant improvements, with relative cpWER (or cpCER) reductions of 29.1%, 19.5%, and 19.7% on LibriCSS, AMI, and AliMeeting, respectively.

5.3. Which factors are most important for GSS?

We performed ablation studies to investigate the effect of several GSS parameters — WPE, noise class, context duration, number of iterations for CACGMM inference, and number of input channels — on the downstream ASR performance, as shown in Fig. 3. WPE was found to be more important for LibriCSS, while using an additional noise class was more important for AMI (Fig. 3(a)). This may be because AMI contains occassional background noise, which is absent in LibriCSS. Increasing the context duration from 5s to 15s resulted in consistent WER gains, but adding further context degraded WER (Fig. 3(b)). We can attribute this to inclusion of the target speaker segments in the context if it is expanded too far. A similar observation was made earlier for CHiME-5 [48], where a 15s context resulted in better WER compared to a 2s context [17].

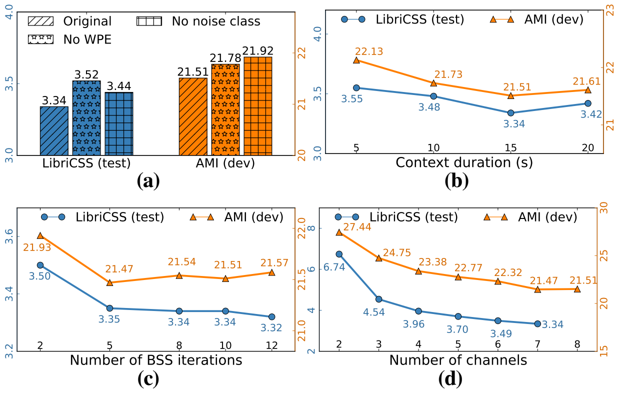

Figure 3: *Impact of several factors on ASR performance. In each figure, the left and right y-axes denote WERs for LibriCSS (test) and AMI (dev), respectively, with the axes scaled according to the range of corresponding WER values.*

Table 4: *Compute time for our GSS implementation compared with original on CHiME-6 dev set, using all available channels, 15s context, and 20 BSS iterations. "Time" is the actual wall clock time (in hours), while "cum. time" is the effective total time for all jobs. Speedup is the ratio of the cumulative times.*

GSS	Compute	Time	Cum. time	Speedup	WER	+RNNLM
Original	80 x Xeon	19.3	1542.6	1.0	44.7	43.5
Ours	4 x V100	1.3	5.3	292.2	44.2	43.1

For both datasets, increasing the number of BSS iterations (for CACGMM inference) beyond 5 did not result in any WER improvements (Fig. 3(c)). Finally, **using more input channels was found to be the single most important factor** for better WER performance. For example, using seven input channels resulted in relative WER reduction of 50.4% and 21.8% on LibriCSS and AMI, respectively, compared to using two channels. Nevertheless, this improvement follows the law of diminishing returns, as evident by the exponential decay in Fig. 3(d).

5.4. Analysis of speed-up

We compared our GSS implementation with the original GSS on the CHiME-6 development set in terms of wall clock time and ASR performance, as shown in Table 4. For ASR inference, we used the publicly available Kaldi recipe and pretrained models from JHU-CLSP's submission to the CHiME-6 challenge [20]. We found that our implementation obtained an **effective speed-up of 292.2** without any degradation in WER. Since CHiME-6 has non-stationary speakers, we disabled segment batching for this experiment. We can obtain even further speed-ups by enabling this for meeting-like data where speakers are stationary.

6. Conclusion

We described our GPU-accelerated implementation of GSS-based enhancement for meeting transcription. On the CHiME-6 benchmark, it was found to be ∼300x faster than the original implementation, thus removing the computational bottleneck associated with this technique. Through experiments conducted on LibriCSS, AMI, and AliMeeting, we showed that GSS can recover up to 80% of the WER difference between close-talk and far-field settings. Ablation studies demonstrated that the number of input channels is the single most important factor determining GSS performance.

Acknowledgment. This project was partially funded by NSF CCRI Grant No. 2120435, and a fellowship from Amazon via the JHU-Amazon Initiative for Interactive Artificial Intelligence (AI2AI).

7. References

[1] D. Raj, P. Denisov, Z. Chen, H. Erdogan, Z. Huang, M. He, S. Watanabe, J. Du, T. Yoshioka, Y. Luo, N. Kanda, J. Li, S. Wisdom, and J. R. Hershey, "Integration of speech separation, diarization, and recognition for multi-speaker meetings: System description, comparison, and analysis," in *IEEE SLT*, 2021.

[2] K. Boakye, B. Trueba-Hornero, O. Vinyals, and G. Friedland, "Overlapped speech detection for improved speaker diarization in multiparty meetings," in *IEEE ICASSP*, 2008.

[3] L. Bullock, H. Bredin, and L. P. García-Perera, "Overlap-aware diarization: Resegmentation using neural end-to-end overlapped speech detection," in *IEEE ICASSP*, 2020.

[4] N. Kanda, Y. Gaur, X. Wang, Z. Meng, and T. Yoshioka, "Serialized output training for end-to-end overlapped speech recognition," in *InterSpeech*, 2020.

[5] W. Chen, W. Hendrix, and N. Samatova, "The application of the weighted k-partite graph problem to the multiple alignment for metabolic pathways," *Journal of computational biology : a journal of computational molecular cell biology*, vol. 24 12, 2017.

[6] J. Yu, B. Wu, R. G. S.-X. Z. L. C. Y. X. M. Yu, D. Su, D. Yu, X. Liu, and H. M. Meng, "Audio-visual multi-channel recognition of overlapped speech," in *InterSpeech*, 2020.

[7] Y. Luo and N. Mesgarani, "Conv-tasnet: Surpassing ideal time–frequency magnitude masking for speech separation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, 2019.

[8] C. Subakan, M. Ravanelli, S. Cornell, M. Bronzi, and J. Zhong, "Attention is all you need in speech separation," in *IEEE ICASSP*, 2021.

[9] J. Cosentino, M. Pariente, S. Cornell, A. Deleforge, and E. Vincent, "Librimix: An open-source dataset for generalizable speech separation," *ArXiv*, 2020.

[10] T. Menne, I. Sklyar, R. Schlüter, and H. Ney, "Analysis of deep clustering as preprocessing for automatic speech recognition of sparsely overlapping speech," in *InterSpeech*, 2019.

[11] Z. Chen, T. Yoshioka, L. Lu, T. Zhou, Z. Meng, Y. Luo, J. Wu, and J. Li, "Continuous speech separation: Dataset and analysis," in *IEEE ICASSP*, 2020.

[12] S. Chen, Y. Wu, Z. Chen, J. Li, C. Wang, S. Liu, and M. Zhou, "Continuous speech separation with conformer," in *IEEE ICASSP*, 2021.

[13] R. Scheibler and N. Ono, "Fast and stable blind source separation with rank-1 updates," *IEEE ICASSP*, 2020.

[14] K. Saijo and R. Scheibler, "Spatial loss for unsupervised multi-channel source separation," in *InterSpeech*, 2022.

[15] 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 *IEEE ICASSP*, 2021.

[16] R. Ikeshita, N. Ito, T. Nakatani, and H. Sawada, "A unifying framework for blind source separation based on a joint diagonalizability constraint," in *EUSIPCO*, 2019.

[17] C. Boeddeker, J. Heitkaemper, J. Schmalenstroeer, L. Drude, J. Heymann, and R. Haeb-Umbach, "Front-end processing for the chime-5 dinner party scenario," in *The 6th CHiME Workshop*, 2018.

[18] N. Kanda, C. Böddeker, J. Heitkaemper, Y. Fujita, S. Horiguchi, K. Nagamatsu, and R. Häb-Umbach, "Guided source separation meets a strong asr backend: Hitachi/paderborn university joint investigation for dinner party asr," in *InterSpeech*, 2019.

[19] S. Watanabe, M. Mandel, J. Barker, and E. Vincent, "Chime-6 challenge: Tackling multispeaker speech recognition for unsegmented recordings," *The 6th CHiME workshop*, 2020.

[20] A. Arora, D. Raj, A. S. Subramanian, K. Li, B. Ben-Yair, M. Maciejewski, P. Żelasko, P. García, S. Watanabe, and S. Khudanpur, "The JHU multi-microphone multi-speaker ASR system for the CHiME-6 challenge," in *The 6th CHiME workshop*, 2020.

[21] H. Chen, P. Zhang, Q. Shi, and Z. Liu, "Improved guided source separation integrated with a strong back-end for the chime-6 dinner party scenario," in *InterSpeech*, 2020.

[22] I. Medennikov, M. Korenevsky, T. Prisyach, Y. Y. Khokhlov, M. Korenevskaya, I. Sorokin, T. Timofeeva, A. Mitrofanov, A. Andrusenko, I. Podluzhny, A. Laptev, and A. Romanenko, "The stc system for the chime-6 challenge," in *The 6th CHiME workshop*, 2020.

[23] P. Comon and C. Jutten, *Handbook of Blind Source Separation: Independent Component Analysis and Applications*. Elsevier, 2010.

[24] S. Horiguchi, Y. Fujita, and K. Nagamatsu, "Block-online guided source separation," in *IEEE SLT*, 2021.

[25] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019.

[26] T. Nakatani, T. Yoshioka, K. Kinoshita, M. Miyoshi, and B.-H. Juang, "Blind speech dereverberation with multi-channel linear prediction based on short time fourier transform representation," in *IEEE ICASSP*, 2008.

[27] ——, "Speech dereverberation based on variance-normalized delayed linear prediction," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 18, 2010.

[28] N. Ito, S. Araki, and T. Nakatani, "Complex angular central gaussian mixture model for directional statistics in mask-based microphone array signal processing," in *EUSIPCO*, 2016.

[29] M. Souden, J. Benesty, and S. Affes, "On optimal frequency-domain multichannel linear filtering for noise reduction," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 18, 2010.

[30] H. Erdogan, J. R. Hershey, S. Watanabe, M. I. Mandel, and J. L. Roux, "Improved mvdr beamforming using single-channel mask prediction networks," in *InterSpeech*, 2016.

[31] C. R. Harris *et al.*, "Array programming with numpy," *Nature*, vol. 585, 2020.

[32] R. Okuta, Y. Unno, D. Nishino, S. Hido, and C. Loomis, "Cupy: A numpy-compatible library for nvidia gpu calculations," in *LearningSys Workshop at NIPS*, 2017.

[33] D. Smith and J. Gray, "opt_einsum - a python package for optimizing contraction order for einsum-like expressions," *Journal of Open Source Software*, vol. 3, 2018.

[34] P. Żelasko, D. Povey, J. Trmal, and S. Khudanpur, "Lhotse: a speech data representation library for the modern deep learning ecosystem," in *NeurIPS Data-Centric AI Workshop*, 2021.

[35] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," in *IEEE ICASSP*, 2015.

[36] J. Carletta, S. Ashby, S. Bourban, M. Flynn, M. Guillemot, T. Hain, J. Kadlec, V. Karaiskos, W. Kraaij, M. Kronenthal, G. Lathoud, M. Lincoln, A. L. Masson, I. McCowan, W. Post, D. Reidsma, and P. Wellner, "The AMI meeting corpus: A pre-announcement," in *MLMI*, 2005.

[37] F. Yu, S. Zhang, Y. Fu, L. Xie, S. Zheng, Z. Du, W. Huang, P. Guo, Z. Yan, B. Ma, X. Xu, and H. Bu, "M2met: The ICASSP 2022 multi-channel multi-party meeting transcription challenge," in *IEEE ICASSP*, 2022.

[38] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang, "Conformer: Convolution-augmented transformer for speech recognition," in *InterSpeech*, 2020.

[39] D. Povey, https://github.com/k2-fsa/icefall/blob/master/egs/librispeech/ASR/pruned_transducer_stateless7/zipformer.py.

[40] D. Snyder, G. Chen, and D. Povey, "Musan: A music, speech, and noise corpus," *ArXiv*, vol. abs/1510.08484, 2015.

[41] F. Kuang, L. Guo, W. Kang, L. Lin, M. Luo, Z. Yao, and D. Povey, "Pruned RNN-T for fast, memory-efficient asr training," in *InterSpeech*, 2022.

[42] W. Kang, L. Guo, F. Kuang, L. Lin, M. Luo, Z. Yao, X. Yang, P. Żelasko, and D. Povey, "Fast and parallel decoding for transducer," *ArXiv*, 2022.

[43] T. J. Park, K. J. Han, M. Kumar, and S. S. Narayanan, "Auto-tuning spectral clustering for speaker diarization using normalized maximum eigengap," *IEEE Signal Processing Letters*, vol. 27, pp. 381–385, 2020.

[44] D. Raj, Z. Huang, and S. Khudanpur, "Multi-class spectral clustering with overlaps for speaker diarization," in *IEEE SLT*, 2021.

[45] H. Bredin and A. Laurent, "End-to-end speaker segmentation for overlap-aware resegmentation," in *InterSpeech*, 2021.

[46] F. Landini, J. Profant, M. Díez, and L. Burget, "Bayesian HMM clustering of x-vector sequences (VBx) in speaker diarization: theory, implementation and analysis on standard tasks," *Computer, Speech, and Language*, vol. 71, 2022.

[47] I. Medennikov, M. Korenevsky, T. Prisyach, Y. Y. Khokhlov, M. Korenevskaya, I. Sorokin, T. Timofeeva, A. Mitrofanov, A. Andrusenko, I. Podluzhny, A. Laptev, and A. Romanenko, "Target-speaker voice activity detection: a novel approach for multi-speaker diarization in a dinner party scenario," in *InterSpeech*, 2020.

[48] J. Barker, S. Watanabe, E. Vincent, and J. Trmal, "The third 'CHiME' speech separation and recognition challenge: Dataset, task and baselines," in *IEEE ASRU*, 2015.