Enhancing audio noise reduction with parallel processing

Zachary Burkett
Undergrad Student, Dept. Computer Science
University of Central Florida
Orlando, USA
za971117@ucf.edu

Joseph Zalusky
Undergrad Student, Dept
Computer Science
University of Central Florida
Orlando, Fl, USA
josephzalusky@ucf.edu

Andrew Bulatao
Undergrad Student, Dept
Computer Science
University of Central Florida
Orlando, Fl, USA
an372391@ucf.edu

***Abstract*—Audio signals often contain unwanted noise, ranging from background sounds to electrical interference. Removing this noise while preserving the desired audio is a computationally intensive task. In this study, we use parallel processing to enhance noise reduction, with the objective of achieving significant performance improvements. Our approach focuses on implementing the spectral subtraction algorithm and parallelizing its two key phases: the analysis of the original audio signal and the application of noise reduction. By utilizing the Fast Fourier Transform (FFT) for frequency analysis and processing the audio in time-chunked segments, we optimize computational efficiency.**

***Index Terms*—parallel processing, noise reduction, Fast Fourier Transform, c++**

I. Introduction

As the need for high-quality audio processing grows across various applications such as telecommunications, entertainment, and assistive technologies, effective noise reduction methods have become increasingly important. In fields such as these, clear audio is crucial for communication, content production, and accessibility. Unwanted noise can stem from various sources, such as background noise and electrical interference, and is often an inevitable part of audio recordings. Traditional noise reduction techniques, such as spectral subtraction, aim to reduce unwanted noise while preserving the integrity of the desired audio [1].

The field of audio noise reduction has seen numerous advancements, with algorithms that improve the accuracy and efficiency of noise suppression. Some are done in real time. Spectral subtraction is a widely adopted technique because of its simplicity and effectiveness in reducing stationary noise [2]. It works by estimating the noise spectrum from the noisy signal and subtracting it from the observed spectrum, leaving a cleaner signal.

This research aims to address the computational limitations of spectral subtraction by using parallel processing. By distributing the workload across multiple processing unites, we can expect reduce processing time and make noise reduction more feasible for large-scale applications.

In live broadcasting, even minor processing delays can result in audio desynchronization or broadcast delays, disrupting the viewer experience. Similarly, in large-scale editing environments, prolonged processing times can hinder workflow efficiency, delaying content delivery and increasing computational overhead. Post-production teams working with extensive audio tracks require swift noise reduction to maintain productivity and meet deadlines.

However, by leveraging parallel computing resources, such as multi-core CPUs, spectral subtraction can be optimized for faster execution, making noise reduction more practical for large-scale or cloud-based workflows. This scalability allows for efficient processing across different hardware architectures, ensuring adaptability for various applications. Through this approach, we aim to enhance the effectiveness of noise suppression in fields like broadcasting and post-production audio editing.

II. Methodology

A. Overview

This study implements a parallelized spectral subtraction algorithm to improve the efficiency of audio noise reduction. Our approach consists of three main stages: (1) processing the audio signal, (2) performing noise estimation and spectral subtraction, and (3) reconstructing the enhanced signal. Using parallelism, we aim to enhance computational efficiency and reduce processing time.

B. Dataset Generation

Raw samples from The Microsoft Scalable Noisy Speech Dataset (MS-SNSD) were used to create the test files [**?**]. Our test files can be found here. Raw noise data, such as a loop of air conditioning running, was overlayed onto a 30 minute speech file with a signal to noise ratio (SNR) of 10, which is considered a moderate to high SNR. This was calculated with the following equation:

$$\text{SNR}(dB) = 20 * log_{10}(\frac{\text{RMS of speech}}{\text{RMS of noise}}) \quad (1)$$

where RMS of speech/noise represents the average power (volume level) of the signal. Therefore, an SNR of 10 represents that the voice signal is 10 times greater than the noise.

These files were created by hand in a digital audio workstation, with benchmark times of 5, 10 and 30 minutes.

C. Data Acquisition and Preproccessing

We use WAV files as input audio signals, which are seperated by channel for independent processing. As the WAV format interleaves channel data, the signal information must first be deinterleaved into seperate channels. The signals are then normalized, and the maximum amplitude from the original audio is saved for use in later reconstruction. The audio is then segmented into overlapping frames to be processed in frequency-domain analysis. It is required to have our analysis frames overlap, as data near the edges of the frame will be lost after windowing.

Each frame undergoes windowing using a Hamming window [3] to minimize spectral leakage during the transformation process. Spectral leakage would lead to distortion and "musical noise" becoming present after subtraction. The Hamming window also ensures that the signal being analyzed by the fast Fourier transform is periodic, which is a requirement for discrete Fourier analysis. The Hamming window function is defined as:

$$w(n) = 0.54 - 0.46 \cdot \cos\left(\frac{2\pi n}{N-1}\right) \tag{2}$$

where n is the index of the sample within the window, and N is the total number of samples in the frame.

D. Spectral Analysis Using FFT

Each audio frame is transformed from the time domain to the frequency domain using the Fast Fourier Transform (FFT). This step is parallelized by distributing the computation across multiple threads, ensuring that frames are processed concurrently. The FFT is applied as follows:

$$X(f) = \sum_{n=0}^{N-1} x(n) \cdot e^{-2\pi i \frac{fn}{N}} \tag{3}$$

where $X(f)$ represents the frequency-domain representation of the signal, $x(n)$ is the time-domain sample, N is the total number of samples, and f is the frequency index.

The noise profile is estimated from the initial segment of the audio signal, assuming the presence of stationary noise. The noise spectrum $N(f)$ is derived from these initial frames, and the noise estimate is applied uniformly across all frames

E. Parallelized Noise Reduction

The spectral subtraction algorithm is applied to each frame of the audio signal. In the frequency domain, it removes the estimated noise from the noisy signal. A threshold is then applied to make sure that no negative values are produced, which helps prevent unwanted sounds (called musical noise) from appearing. The formula for spectral subtraction is given by:

$$\hat{X}(f) = \max(X(F) - (N(f), 0) \tag{4}$$

where $\hat{X}(F)$ is the enhanced signal spectrum, $X(f)$ is the original noisy spectrum, and $N(f)$ is the estimated noise spectrum.

We implement a threshold mechanism to avoid subtracting more than the actual noise, ensuring the magnitude is non-negative:

$$\hat{S}(f) = \max\left(|X(f)| - |N(f)|, 0\right) \tag{5}$$

where $|X(f)|$ and $|N(f)|$ represent the magnitudes of the noisy signal and noise spectra, respectively.

F. Signal Reconstruction

After subtracting the spectra of the noise, the data must be converted back into the time domain to output as a pulse code modulation based .wav file. This requires running each of the frames through the inverse fast Fourier transform and summing them back into the entire output signal. Since the frames were windowed with the Hamming function, this will have to be undone for each frame in order to reconstruct the original signal. Furthermore, as frames have overlap, the total weight from the Hamming functions of both overlapping frames will have to be calculated to undo the windowing.

After calculating the weights used across the overlapping frames, we iterate over the frames and divide by the calculated weight to recover the original signal [4]. This process is known as the weighted overlap add algorithm [5]. After the frames have been added together, we denormalize the audio using the previously calculated maximum amplitude in order to preserve the original volume of the audio. The signal is also reinterleaved for the amount of channels from the original input file.

III. Implementation

A. Overview

The implementation of our noise reduction systems focuses on optimizing the spectral subtraction algorithm through multi-threaded processing. Our workflow is divided into preprocessing, noise analysis, noise reduction, and audio reconstruction. Each part of these workflows use parallel computation where it is beneficial to enhance efficiency.

B. Preprocessing

As mentioned in the methodology, we first deinterleave the separate channels of the input audio signal (a WAV file) into seperate channels. This is represented as a 2D array in the code like this:

```
int16_t samples[channels][samples]
```

The signal is then normalized and split into overlapping frames, with an overlap ratio of 50%. Then, we apply the Hamming window to each frame.

This operation is generally quite fast, and the overhead of threading this for multiple channels would likely not be worth the overhead of spawning and retrieving data from multiple threads. Furthermore, the typical audio file only has 1-2 channels, so parallelization across channels would only have small gains.

C. *Noise Analysis*

We assume the stationary noise is contained within the first N frames of the audio file. By default, we use 50 frames to calculate the profile, but more can be specified by the user. The magnitudes of these frequency spectra (calculated by executing an FFT on the first N frames) were averaged to create a channel-specific noise profile. This step is parallelized across each channel, where a separate thread handles each channels noise estimation.

D. *Parallelized Noise Reduction*

The core of the noise reduction algorithm operates on the overlapping frames for each channel along with it's corresponding noise profile. For each frame in a channel, the estimated noise spectrum is subtracted from the noisy signal's spectrum in order to reduce the noise.
Since the actions can independently be performed on each frame, we can distribute this workload across threads. We parallelize this workload by "chunking" each channel's frames into smaller subsets of size 32. After this, we can submit the spectral subtraction work into a thread pool, where each thread will handle spectral subtraction across a chunk of frames. Furthermore, since the overlap add (the process to convert frames back to samples) and deweighting of each sample (reversig the Hamming window) is independent, the individual threads can also handle this workload and return back a chunk of noise-reduced samples.

E. *Audio Reconstruction*

Since the threads will each return their own chunk of the resulting noise-reduced samples for each channel, the samples must be flattened back into a single one-dimensional array per channel.
Then, we need to convert our samples back to `int16_t` from `double`, as needed by the WAV standard, as our signal processing steps (i.e the FFT) operated on the set of real numbers. We also denormalize our samples and clamp them to the range of `int16_t`.

IV. Evaluation

A. *Overview*

Benchmarks on a 12-core CPU against the 30 minute sample file showed a processing time of 3.95 seconds compared to 11.62 seconds using sequential execution, showcasing a near 3x performance speedup.

Memory usage increased by about 20% when using parallel execution.

Acknowledgment

We would like to acknowledge Benji for giving us the motivation and morale to complete this project.

References

[1] S. V. Vaseghi, "Spectral subtraction," *Advanced Signal Processing and Digital Noise Reduction*, pp. 242–260, 1996.
[2] Columbia.edu, 2025. [Online]. Available: https://www.columbia.edu/~ahk2149/DSP_Project/spectral_subtraction.html
[3] D. F. Elliott, *Handbook of digital signal processing : engineering applications*. Academic Press, 1987.
[4] W. Contributors, "Overlap–add method," Wikipedia, 05 2024.
[5] "The overlap-add method," Dspguide.com, 2025. [Online]. Available: http://www.dspguide.com/ch18/1.htm