GPU Filter Banks for Audio

Dan Jacobellis and Utsha Khondkar

Department of Electrical and Computer Engineering
University of Texas at Austin,
Austin, TX 78712
{danjacobellis, utsha.kh}@utexas.edu

Abstract. In our project, we implement a class of perfect-reconstruction filterbanks for audio analysis. We use a GPU to solve embarassingly parallel subtasks so that the algorithm takes only $O(\log N)$ time steps. We also introduce a novel type of time-frequency tiling which retains the benefits of constant-Q transforms without requiring their cumbersome data structure.

1 Background

For audio signal processing problems such as compression, denoising, and various types of pattern recognition, deep learning has taken over. For example, the search interest for "wavelet" has followed the opposite trajectory as "convolutional neural network"

A current trend of audio signal processing research is to apply 2D convolutional neural networks to time-frequency representations of audio. Although great progress was made in the pre-deep learning era on perfect reconstruction filterbanks and wavelet transforms, most of these algorithms and techniques have been abandoned by practitioners of deep learning because no standardized and free implementations were ever widely disseminated. As a result, the standarized set of triangular "Mel" filterbanks dating back nearly 50 years appears to have remained the dominant tool, and have even seen a resurgance despite their impending obsolescence. [Geo+21]

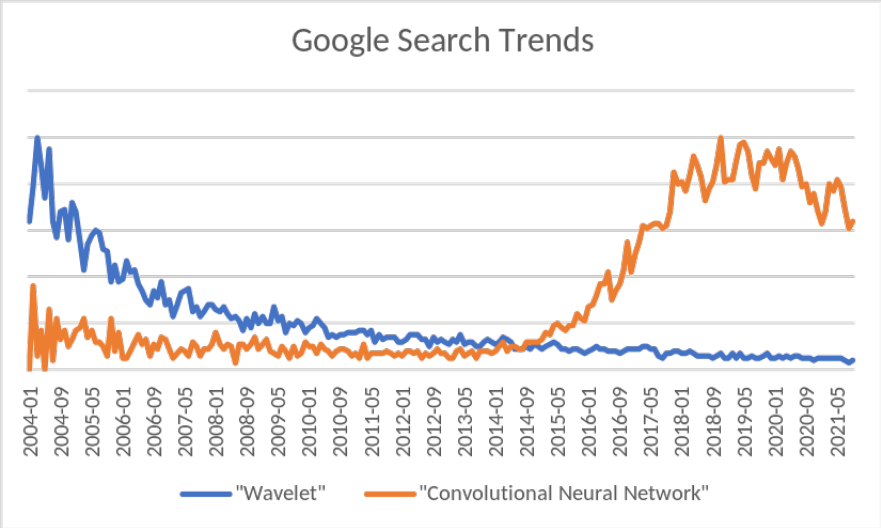


Fig. 1: Some expository search trends

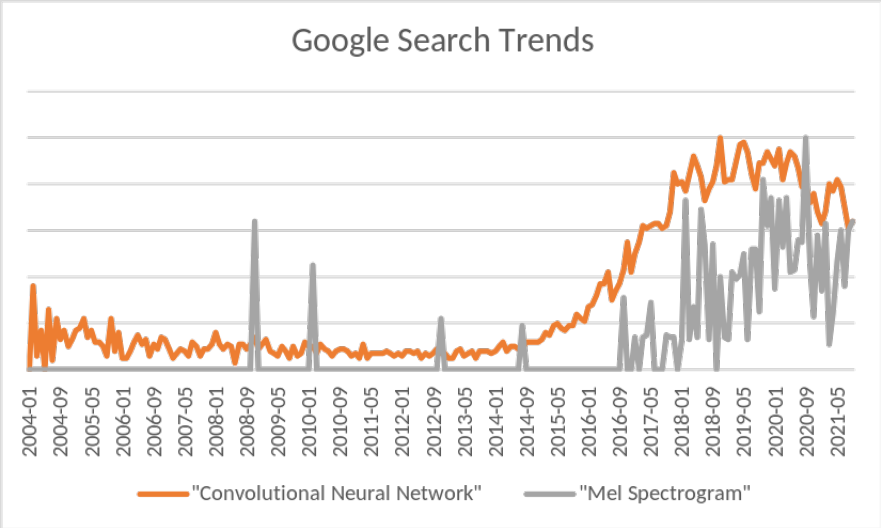


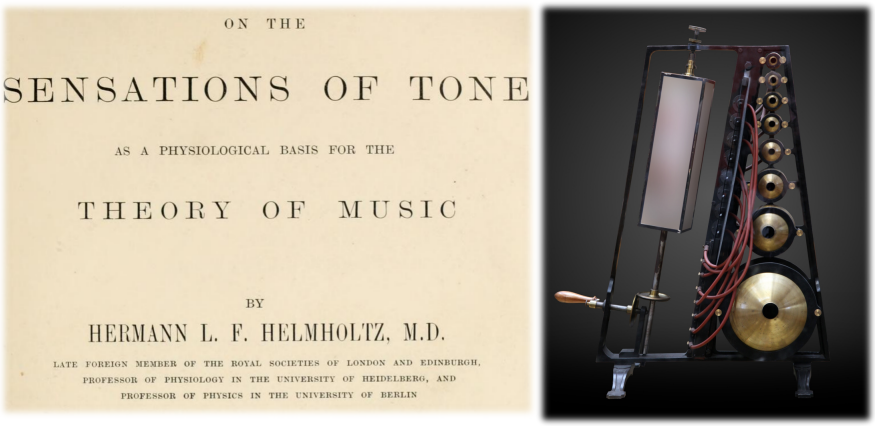
Fig. 2: More expository search trends

The use of these methods is accumulating a considerable debt. Because they are based on windowed FFTs, the time resolution is the same for all frequencies. This causes the representation to be overly redundant for low frequencies while undersampled for high frequencies. As a result, they lack the perfect re-

construction property and have an inconvenient phase representation. They are difficult to generalize to multiple channels, phased arrays, or higher dimensional transforms.

1.1 Filter banks and discrete wavelet transforms

Filter banks are the oldest method of time-frequency analysis, dating back to the first analog spectrum analyzer built by Hermann von Helmholtz in the 19th century.


ON THE

SENSATIONS OF TONE

AS A PHYSIOLOGICAL BASIS FOR THE

THEORY OF MUSIC

BY

HERMANN L. F. HELMHOLTZ, M.D.

LATE FOREIGN MEMBER OF THE ROYAL SOCIETIES OF LONDON AND EDINBURGH,
PROFESSOR OF PHYSIOLOGY IN THE UNIVERSITY OF HEIDELBERG, AND
PROFESSOR OF PHYSICS IN THE UNIVERSITY OF BERLIN

Fig. 3: Early work on analog time-frequency analysis

In 1909, mathematician Alfréd Haar discovered that a list of 2^n numbers can be represented by recursively taking the sum and difference of adjacent pairs. This property is now commonly referred to 'alias cancellation'. In the 1970s, engineers created the first discrete, invertible filterbanks by expoiting this property. Using what are known as 'conjugate mirror filters'. In the following decade, a massive research effort took place, resulting in a rigorous mathematical understanding of these processes and the development of various "wavelet transforms."

1.2 Haar Decomposition

The Haar decomposition is a simple type of perfect reconstruction filter bank and is the basis of our algorithm. It used the shortest possible filter high-pass and low-pass filters with perfect alias cancellation: The two tap average and the first order difference. Since the downsampling factor is equal to the number of filters, it is critically sampled. Each application of this block doubles the frequency resolution while halving the time resolution. The inverse operation is also achieved by a two-tap average and first order difference. Recursively application of this procedure is a simple type of discrete wavelet transform. The same procedure could be applied using other quadrature mirror filters based on the desired properties of the transform,[Mal08] but for simplicity we have limited ourselves to the Haar filters.

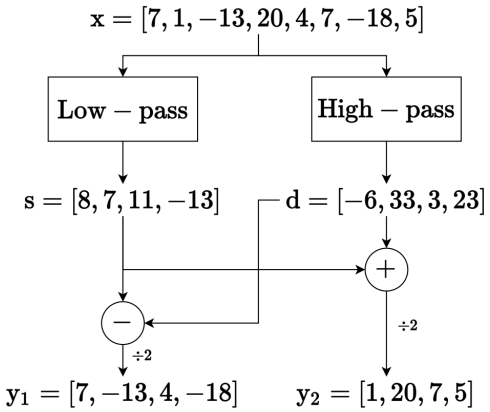


Fig. 4: Single stage of Haar filter bank

2 Algorithm

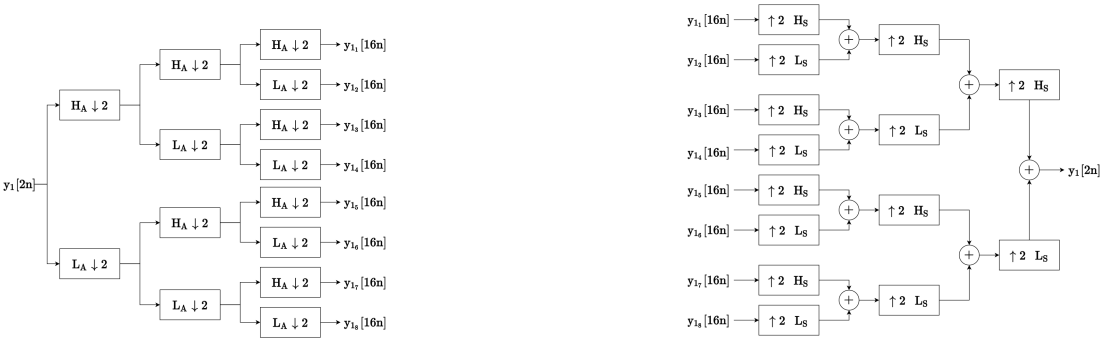
Discrete wavelet transforms are highly parallelizable. However, the data structure that results from recursive decomposition of the low-pass component is

cumbersome, especially if we want to perform other types of processing on the resulting time-frequency distribution. As a result, we have chosen to combine to two common dyadic filterbank structures to achieve nearly constant-Q frequency resolution while representing each octave's time-frequency representation with a single matrix.

2.1 Recursive assymetric/symmetric tree

(a) Analysis (b) Synthesis

Fig. 5: Assymetric Filterbanks



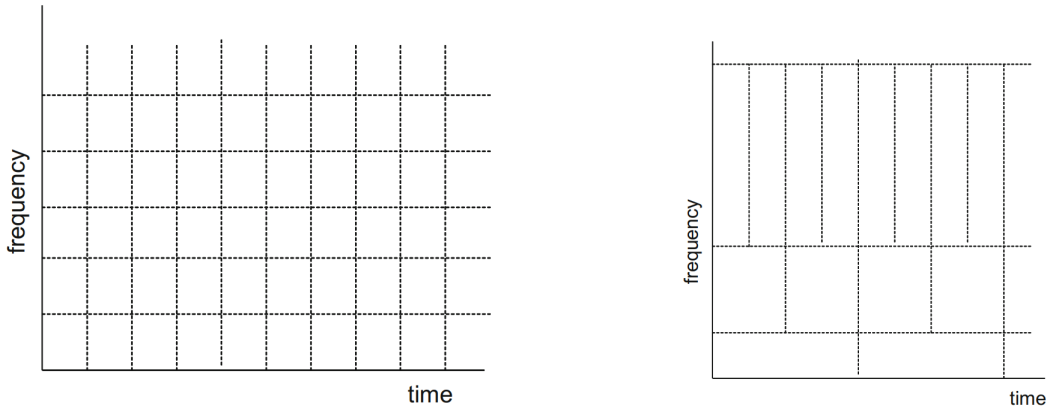
(a) Analysis (b) Synthesis

Fig. 6: Symmetric Filterbanks

In our algorithm, we recursively apply analysis filters to decompose the low frequency band until we have 10 octaves of resolution (plus a DC band). Then, each octave is further refined in a second stage using the same dyadic filter bank,

but this type we recurse for the high-pass component as well. The synthesis filter bank follows the same structure to perform the inverse operation. Since each decomposition reduces the time resolution by a factor of two, there can be a maximum of $\log N$ levels in the tree.

2.2 Tiling of hybrid approach



(a) STFT (b) Constant-Q

Fig. 7: Tilings

The output of the first stage creates a tiling of the time-frequency plane into octave bands. This provides greater frequency resolution near the low end of the spectrum, but greater time resolution for high frequencies, which is useful for many applications. The output of the second stage further refines each octave into uniformly sized rectangular time-frequency tiles which can be stored and processed conveniently as a matrix.

3 Implementation

Our implementation, along with examples demonstrating the perfect reconstruction property and a simple denoising application are available at danjacobellis.github.io/GPUfilterbank. We used the 'CUDA' package in Julia [BFD19] to access the highly optimized CuDNN convolution routines. [Che+14]

3.1 CuDNN Convolution

The computation required to compute the convolution can be expressed in terms of a matrix vector product of a Toeplitz matrix containing the signal x and a vector of the filter coefficients h.

$$y = x * h = \begin{bmatrix} x_1 & 0 & \cdots & 0 & 0 \\ x_2 & x_1 & & \vdots & \vdots \\ x_3 & x_2 & \cdots & 0 & 0 \\ \vdots & x_3 & \cdots & x_1 & 0 \\ x_{n-1} & \vdots & \ddots & x_2 & x_1 \\ x_n & x_{n-1} & & \vdots & x_2 \\ 0 & x_n & \ddots & x_{n-2} & \vdots \\ 0 & 0 & \cdots & x_{n-1} & x_{n-2} \\ \vdots & \vdots & & x_n & x_{n-1} \\ 0 & 0 & 0 & \cdots & x_n \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ h_1 \\ h_2 \\ \vdots \\ h_m \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

For a single sample, convolution can be thought of as a dot product between the signal and the filter coefficients. Fortunately, the dot product required to compute each output sample can be executed independently from all other samples. The only dependency preventing the problem from being 'embarrassingly' parallel is that the summation requires a scan. However, for a filter of length M, this step only requires $\log M$ steps assuming that the maximum number of processors $p = M$ are utilized. Exploiting the highly optimized CuDNN convo-

lution routines allows us to get as close as possible to this theoretical bound. If the number of processors is increased to $p = NM$, then the entire convolution can be completed in only $\log M$ time steps.

3.2 Performance bounds and memory considerations

With unlimited processors, our algorithm would complete in $(\log N)(\log M)$ time steps. However, this is not realistic considering that sizes of audio data can easily exceed 10^{10} samples for large datasets. In practice, we are limited by the ability of CPU and data storage to feed data to GPU. However, by keeping a constant stream of data, we can hope to make p a large as possible so that data can be processed in the minimum number of batches.

4 Results

We tested how our algorithm scales with different data sizes, from about one second of music to one hour of film. Even when the data size approached the capacity of the GPU, we were able to complete the full transform in roughly one second. The results shown in Fig. 8 are the median of five trials. The synthesis filter bank is only moderately more expensive than the analysis filter bank.

Next, we compared our algorithm with the mel spectrogram implemented in Librosa, which is a widely-used audio processing library. Since the time-resolution is fixed, it is only capable of approximate reconstruction using a phase-reconstruction algorithm such as like Griffin-Lim.

References

[BFD19] Besard, T., Foket, C., and De Sutter, B. “Effective Extensible Programming: Unleashing Julia on GPUs”. In: *IEEE Transactions on*

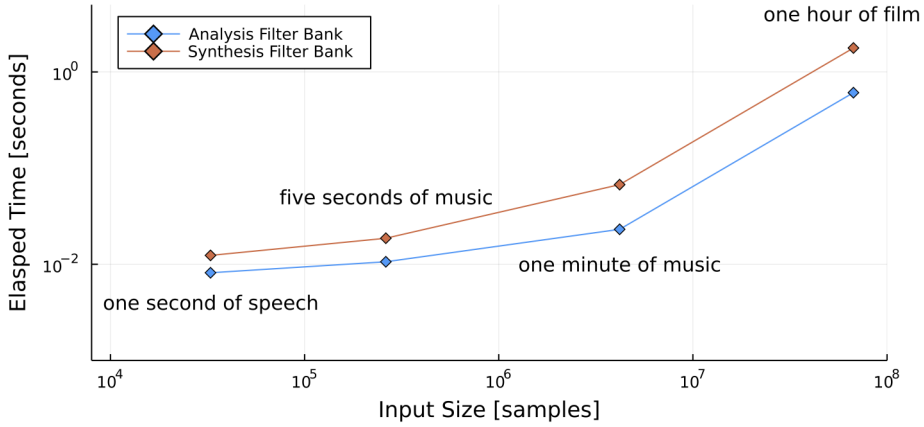


Fig. 8: Performance measurement of our algorithm

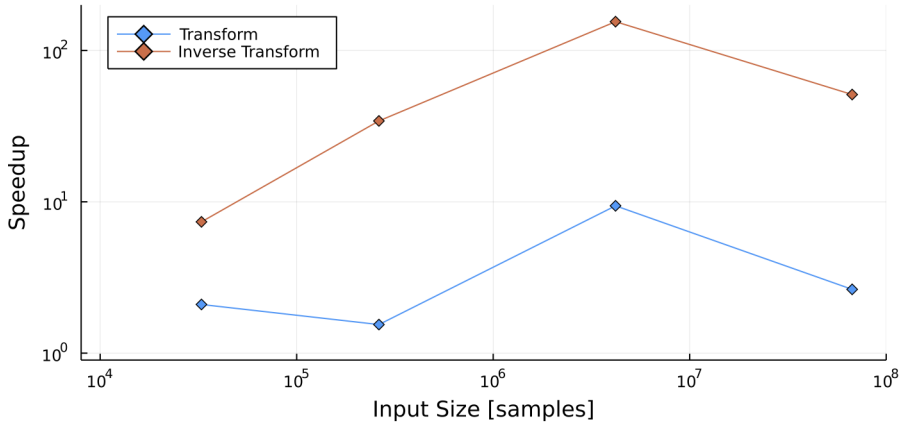


Fig. 9: Speedup compared to librosa mel spectrogram

Parallel and Distributed Systems 30.4 (2019), pp. 827–841. DOI: 10.1109/TPDS.2018.2872064.

[Che+14] Chetlur, S. et al. "cuDNN: Efficient Primitives for Deep Learning". In: (Oct. 2014).

[Geo+21] Georgescu, A.-L. et al. “Performance vs. hardware requirements in state-of-the-art automatic speech recognition”. In: *EURASIP Journal on Audio, Speech, and Music Processing* 2021 (2021), pp. 1–30.

[Mal08] Mallat, S. *A Wavelet Tour of Signal Processing: The Sparse Way*. Elsevier Science, 2008. ISBN: 978-0-08-092202-7. URL: https://books.google.com/books?id=5qzeLJljuLoC.