Real-Time Channel Vocoder Implemenation

Ajwad F. Choudhury

***Abstract*—A channel vocoder used in musical applications for synthesising human speech is implemented using the programming language C, and its performance is analysed to see if it is suitable for being deployed as a real-time system. For a low decimation/interpolation factors, $M = 8$, a block time of 8.1 ms and 2.7 ms is achieved for encoding and decoding respectively. This increases as M increases but more significantly for decoding, doubling to 4.3 ms for $M = 160$. Different factors are considered as to why this increase in latency occurs and various solutions are suggested to tackle these issues such as memory management and test-benches.**

I. Introduction

Invented in 1936 by Homer Dudley, the channel vocoder serves as a device that encodes human speech signals for applications such as compression and encryption [1]. Modern solutions involve a series of digital finite impulse response (FIR) filters which can be easily implemented on scientific programming languages such as MATLAB or Python with digital signal processing (DSP) frameworks and libraries. However, this leads to large overheads due to their high-level abstraction, impacting run-time. Since modern systems operate on indefinitely long signals, the channel vocoder must be implemented as a real-time system which must perform within specific time constraints, rendering these two languages too slow. Therefore, real-time systems are usually developed using low-level languages such as C or C++ which are optimised to run quickly, and deployed onto real-time/embedded systems such as a micro-controller.

A channel vocoder can be separated into an encoder and decoder. The encoder consists of an array of band-pass filters (BPF) set at unique centre frequencies and bandwidths, each followed by an envelop detector and decimator. The decoder consists of a pseudo-random signal filtered by the same BPF array and multiplied by the interpolated envelop. The sum of all these products form the reconstructed speech signal. The objective of this project was to realise this system using the C language and achieve real-time system characteristics with the lowest latency possible.

II. Methodology

A. FIR Filters

The three types of filters implemented were a BPF, Hilbert filter (HBF) and low-pass filter (LPF). Finite Impulse Response (FIR) filters were used to implement these because of their stability and linear phase, but crucially their ease of implementation. The centre frequencies of each BPF were determined by analysing the spectrum of the input signal to see where the dominant frequencies lied in human speech. This was found to be between 100-1000 Hz, thus the majority of the filters were placed in this region to capture these major frequency components. Implementing the BPF filters involved generating the filter coefficients from the impulse response. A window function was applied to attenuate the Gibbs effect which is shown by the pass-band ripple/oscillation on the magnitude Bode plot of the filter. The kaiser windowing function was used because of its variable stop-band attenuation which is a function of its single parameter, β. The filter order was determined by using a value in the same order of magnitude as the block size of 128. Values between 60-100 were tested and found that anything above 70 caused a large performance deficit. A filter order of 61 was chosen, and to compensate for the lower accuracy, β, was varied to ensure the side-lobes attenuated the frequencies outside the cutoff region by at least -5 dB.

The envelop detector was implemented using the Hilbert transform which expresses the analytical signal, $s_a[n]$, as

$$s_a[n] = A[n]e^{j\theta n} \tag{1}$$

where, $A[n]$ - amplitude, allowing the envelop of the signal to be recovered. To save on initial overhead, the filter coefficients for each of the impulse responses were generated in MATLAB using the filter designer which is part of the signal processing toolbox. The coefficients were then exported as a text file and imported into arrays in the C file, eliminating any unnecessary FOR loops. This was however not possible for the LPF as the cutoff frequency is a function of the decimation factor, therefore was manually generated in the program.

B. Block Convolution

Real-time systems do not perform operations on the entire signal, but instead perform on smaller sized blocks. A block size of 128 samples was chosen as this increases driver efficiency [2]. In order to apply a filter to a signal, a convolution ($*$) between the input signal, $x[n]$, and impulse response of the filter, $h[n]$, must take place. The two main methods of executing a convolution are given by equations (2, 3).

$$x[n] * y[n] = \sum_{m=-\infty}^{\infty} x[m]y[n-m] \tag{2}$$

$$x[n] * y[n] = X(e^{j\theta})Y(e^{j\theta}) \tag{3}$$

Both these equations are valid approaches however, from a time complexity standpoint (2) would take longer to run as it would have to be employed using a second-order nested FOR loop, resulting in a quadratic time complexity $O(n^2)$, for any length n. Conversely, (3) involves computing and multiplying the Discrete Fourier Transforms (DFT) of the input signal

TABLE I
ENCODING PERFORMANCE

	Encoding	
M	Average Block Time (ms)	Total Run Time (s)
8	8.116	19.048
16	8.826	20.174
40	8.941	20.985
80	9.254	21.720
160	9.274	21.766

	Decoding	
M	Average Block Time (ms)	Total Run Time (s)
8	2.693	6.324
16	2.833	6.652
40	4.426	10.392
80	4.142	9.726
160	4.345	9.825

and impulse response. The DFT can be employed using an algorithm called the Fast Fourier Transform (FFT), provided by the FFTW3 C library which runs with linearithmic time complexity $O(n \log n)$ for non-prime values of n [3].

To correctly perform a linear convolution, the previous block must be accounted for by adding its last $P-1$ samples (where, P - Impulse Response Length) to the first N samples of the current convolution (where, N - Input Signal Length). This must be done for the subsequent block convolutions and therefore a 'tail' is needed to store previous convolution values. This algorithm was implemented using the overlap-add method which is demonstrated by [4]. The trade-off with this method is that a store is needed for to keep track of all tails for the BPF, HBF and LPF, increasing the space complexity of the program, which may be unsuitable for memory constrained systems.

III. RESULTS & DISCUSSION

The performance of the program was tested against different decimation/interpolation factors, M, with their corresponding sampling frequencies, f_s. The performance was measured by averaging the run-time for each block process and also measuring the total run-time, and have been tabulated into Table 1.

It can be seen that encoding takes roughly three times longer on average to execute compared to decoding. This is due to the encoding stage having two more convolutions to perform per block. As M increases, the block time for both processes increases, and does so considerably for decoding for very high values, almost doubling the block time from 2 ms to 4 ms. The encoding block time is less-so impacted but still reports an increase by roughly 1 ms. A possible explanation for this could be due to the type of data structure used for storing the values, as widely spaced values inside arrays could be computationally expensive for operations such as searching, accessing and insertion. More efficient data structures may be appropriate for this applications such as binary search trees or hash tables however require object-oriented programming languages such as C++ which is beyond the scope of this project. Other factors could relate to the test bench used for the experiment which was a general purpose computer running a general purpose operating system which handles multiple applications concurrently. This doesn't guarantee full priority and access for the program to the CPU and therefore may be delayed by interrupts and thread deadlock. Additionally, this program was run on a Linux virtual machine which has constrained access to the full resources of the host computer. An ideal test bench would be a system with a real-time operating system used for mission critical applications, having high predictability and rigorous resource management and scheduling [5]. This would typically be deployed onto an embedded system such as a micro controller or field programmable gate array (FPGA) which has programmable logic, requiring no operating system at all.

When analysing the signal fidelity, there is noticeable deficit in clarity of human speech as expected from a vocoder, which is used to synthesise human voice in the music industry. The change in amplitude can be heard but the exact speech cannot be deciphered and is mostly robotic. Increasing M has no audible effect in quality.

IV. CONCLUSION

It has been demonstrated how a channel vocoder has been implemented in C, with initial results showing block times which are small enough for a real-time system. The latency of 8 ms for encoding and 2 ms for decoding on top of the minimum latency of 16 ms would be hardly noticable to the human ear as the mouth-to-ear latency of a phone call is 200 ms [7]. Further work would involve improvements in memory management by eliminating redundant variables and utilising more allocated memory. Furthermore, exploring vectors in C which inherit methods that support matrix multiplication and addition, running with better time complexity compared to the FOR loops implemented.

REFERENCES

[1] US 135416A Patent, "System for the artificial production of vocal or other sounds", issued 1937-04-07

[2] "Block size," www.ibm.com. https://www.ibm.com/docs/en/spectrum-scale/5.1.0?topic=considerations-block-size (accessed Mar. 12, 2022).

[3] "FFTW FAQ - Section 4," www.fftw.org. https://www.fftw.org/faq/section4.htmlwhyfast (accessed Mar. 12, 2022).

[4] "Learn about the Overlap-Add Method: Linear Filtering Based on the Discrete Fourier Transform - Technical Articles," www.allaboutcircuits.com. https://www.allaboutcircuits.com/technical-articles/overlap-add-method-linear-filtering-based-on-the-discrete-fourier-transform/ (accessed Mar. 12, 2022).

[5] "What Is a Real-Time Operating System (RTOS)? — Ultimate Guides — BlackBerry QNX," blackberry.qnx.com. https://blackberry.qnx.com/en/ultimate-guides/what-is-real-time-operating-system: :text=An%20RTOS%20provides%20the%20rigorous (accessed Mar. 12, 2022).

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] "G.114 : One-way transmission time". www.itu.int. Retrieved 2019-11-16