Implementing a Speech Recognition System on a Graphics Processor Unit (GPU) using CUDA

Astrid Yi (996864558), Omid Talakoub (994951307)

I. Introduction

A speech recognition system can be classified based on two factors: (1) whether the system is speaker-dependent or speaker-independent, and (2) whether the system works for continuous speech or isolated words. Ideally, it should be able to recognize each spoken word accurately regardless of the speaker, but this type of system does not exist yet. As a result, the development of a continuous speaker-independent speech recognition system remains an ongoing research goal.

A speech recognition engine can be used in many different applications. For instance, it can help people with learning disabilities, who cannot properly map their thoughts into writing, by generating a written version of their speech. It can also be used in an aircraft to alleviate the pilot's workload by executing the uttered commands.

All of the practical applications rely on the achievement of very high recognition accuracy. Although isolated-word speech recognition systems can only recognize one word at a time, they tend to be the preferred choice since their performance accuracy is higher than that of continuous speech recognizers.

Speech recognition is carried out in two phases. In the training phase, the system memorizes a set of reference templates. In the test phase, the system compares a speech signal with all of the reference templates and returns the closest one as the recognized pattern. It should be noted that performance accuracy improves when there are more reference templates to compare with. However, the time to find the closest match increases exponentially with a larger set of reference templates. This time can be reduced by parallelizing computationally expensive tasks on a graphics processor unit (GPU).

This paper reveals a marked improvement by moving the computationally expensive tasks onto a GPU. In fact, the GPU implementation is about 5.8 times faster than the CPU one.

II. Related works

A few studies attempted to implement a speech recognition algorithm on a graphics processor unit (GPU). Poli et al. (2007) applied the dynamic warp algorithm (DTW) to perform voice password identification, and they reported that it is possible to obtain an increase in performance by moving the computations onto a graphics card [1]. Cardinal et. al (2008) used NVIDIA GeForce 8800 GTX to compute the acoustic likelihoods for their speech recognition system, which is based on a finite-state transducer (FST) framework. They gathered the average CPU and GPU times by computing the acoustic likelihoods 2000 times, and they reported a performance increase of 33.5% with the GPU implementation [2]. Chong et. al (2008) explored the opportunities for parallelizing a more complex speech recognition algorithm. They implemented a hidden Markov model (HMM) based Viterbi search algorithm, which is typically suited for a large-vocabulary continuous speech recognizer. Their GPU version proved to be 9 times faster than their CPU version [3].

III. Dynamic time warping

A. Background

In general, it can be observed that the time scale of a test signal is not perfectly aligned with that of a reference signal. A popular choice for determining the similarity between the test and reference signals involves the computation of the Euclidean distance. However, this method is not always reliable. Consider Figure 1 which shows two sequences of data with the same overall shape but with different time alignment. Indeed, the Euclidean distance does not provide an accurate measure of the distance between the i^{th} point of one sequence and the i^{th} point of the second sequence. Instead, a nonlinear time warping technique needs to be used, as illustrated in Figure 2. An example of such a technique includes the dynamic time warping (DTW) algorithm which is often used in the speech processing community [4].

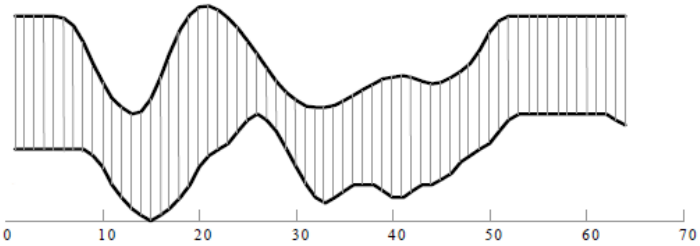

Figure 1: Two sequences from an Australian Sign Language dataset. The Euclidean distance produces a dissimilarity measure [4]

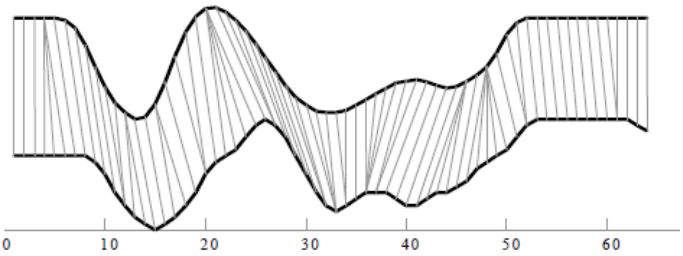

Figure 2: Two sequences from an Australian Sign Language dataset. A nonlinear time alignment allows for a more sophisticated measure of distance [4]

B. *Description of the Algorithm*

Dynamic-time warping (DTW) aims at aligning two sequences of feature vectors by warping the time axis iteratively. In the case of a speech recognition system, a feature vector is derived from a spoken word, and it can have several different representations such as the discrete Fourier transform (DFT) coefficients and the cepstral coefficients, which are obtained from the inverse DFT of the logarithm of the magnitude of the DFT coefficients. This paper resorts to the use of Mel-frequency cepstral coefficients (MFCCs), which represent coefficients of the short-term power spectrum of a sound. Figure 3 shows how these coefficients can be derived. First, the Fourier transform of a windowed signal is taken. Then, the powers of the spectrum are mapped onto the mel scale by using overlapping triangular windows. The logarithms of these powers are computed at each mel frequency, and the corresponding discrete cosine transform (DCT) is obtained. Finally, the high-frequency components of the resulting signals are discarded since they do not provide relevant speech information [5].

Figure 3: Typical derivation of MFCCs

After obtaining the MFCCs, the smallest warping path must be found in order to recognize a word while satisfying the given boundary, continuity, and global constraints. Boundary conditions specify the starting and end points of the warping path while the continuity ones determine the allowable steps in the warping path, and the global conditions define the search region used in finding the optimal path.

IV. Implementation of the Speech Recognition System

The initial development of the speech recognition system was completed in MATLAB. This implementation was subsequently converted to a pure C++ program in order to remove the dependency on MATLAB. The C++ version of the speech recognition system was then used as a reference to create the GPU CUDA based implementation.

Figure 4 gives an overview of the algorithm with regards to how it was implemented in the C++ program. The first step involves loading the sound data from the reference files as well as the test file. A set of MFCCs is generated for each file loaded. To generate the MFCCs, the sound data is first segmented into overlapping frames. For each frame, its associated coefficients are computed. Once all these computations have completed, the test sound data is compared with the references to generate a cost using DTW for each reference. The costs are subsequently sorted in ascending order, and the reference associated with the lowest cost is the recognized word.

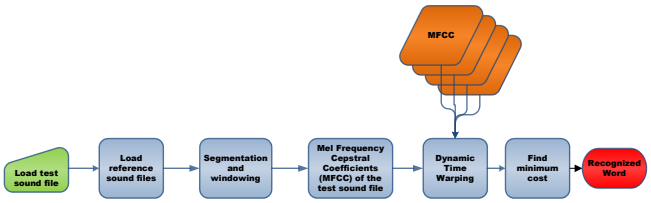

Figure 4: Overview of the speech recognition algorithm

Figure 5 shows how parallelization can be done on a GPU. Indeed, the availability of many processing cores along with the CUDA runtime allow for the simultaneous execution of all the data frames. This cannot easily be done on a CPU due to the limited number of execution units and the need to explicitly launch a thread of execution for each core. As a result, each frame tends to execute sequentially on a CPU.

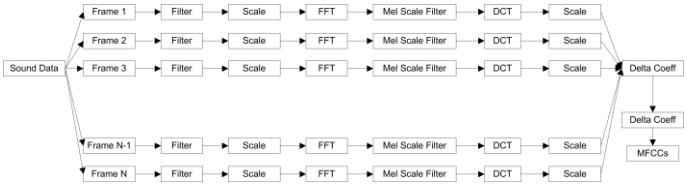

Figure 5: Data flow diagram on the GPU for calculating MFCCs

A. *MFCC Generation*

To generate the MFCCs, the speech signal to be recognized must be partitioned into a set of overlapping frames. Each frame is processed in the same manner to generate a set of coefficients. As each frame receives the exact same processing, this presents a good way to approach the implementation for the generation of the coefficients.

Each processing step on the speech signal frames can be considered as a transformation between the input data and the desired output. In CUDA, the terminology usually associated with the code which performs such a transformation is a kernel. When launching a CUDA kernel, a set of parameters is specified to determine the number of instances that the kernel will execute. If the GPU does not have the necessary resources to run all the desired instances of the kernel concurrently, the CUDA runtime will attempt to schedule a series of kernel invocations in the most efficient way possible. Using the provided scheduling in the CUDA runtime, it is possible to launch a kernel so that there will be a running instance for every frame of the speech signal. If the GPU does not have the necessary processing units to do this simultaneously, the necessary scheduling required is performed automatically.

There are several steps required in the calculation of the coefficients, each of which is implemented in the form of a CUDA kernel. In order to do this, there are six kernels required: infinite impulse response (IIR) filter, scale a vector by a

constant, scale a vector element by element using a second vector to contain the scaling values, a fast Fourier transform (FFT), a mel scale filter, and a discrete cosine transform (DCT). Each of these kernels can process all the data frames at once without the need for multiple explicit invocations. Of these six kernels, the FFT was provided via the CUFFT library while the rest had to be written.

B. *Target Matrix Generation*

The target matrix which is used to compare against all the references to find a match is created using the previously generated MFCCs. The matrix itself has as many rows as there are speech signal frames and three times the number of MFCCs for columns. The first series of columns of the target matrix are populated using the MFCCs while the other columns are populated using the MFCCs which have been transformed by going through a delta operation. The result of the first delta operation is placed in the target matrix directly to the right of the original MFCCs. The result of the first delta is then placed through the same delta operation again to create a second set of data which is placed in the remaining matrix entries. This delta operation was also implemented via a CUDA kernel.

C. *Dynamic Time Warping*

The implementation of the DTW was simple as the core optimization was done one the generation of the MFCCs. To implement the DTW, the code from the C++ implementation was copied over to a CUDA kernel. This kernel could then simultaneously compare the target matrix to all the potential feature matrices.

V. Methodology

In any speech recognition system, there are two important metrics to consider: the elapsed time between the input (acquisition of the speech signal) and output (recognized word), and the recognition accuracy. For the GPU implementation, the accuracy desired was equal to that of the CPU version. However, there were some slight differences (between the GPU and CPU implementations) due to variations in the FFT algorithm and the implementation of the floating point hardware. After comparing the GPU results with the CPU ones, it was found that these small changes were not significant and did not impact the recognition accuracy. Thus, it is not worthwhile to pursue a further discussion on this particular metric.

Since the implementation of the dynamic time warping algorithm is identical to the original one, no change is expected in its performance. Therefore, performance results are only examined for the generation of MFCCs and feature matrices.

CUDA provides a set of utility functions, some of which are designed to act as a timer. However, these utility functions cannot be used to since the CPU version of the speech recognition system does not use CUDA. In order to have consistent performance measurements between the CPU and GPU implementations, it is possible to resort to QueryPerformanceCounter, which is a function from the Windows API.

Performance results were obtained by gathering the amount of time it takes to calculate 45 MFCCs and feature matrices. The calculation of multiple MFCCs and feature matrices helped remove the impact of any potential startup code on the first invocation, which would not occur after initializing the application. The whole process was repeated 5 times, and the results were averaged.

VI. Results

The performance results of the GPU and CPU implementations revealed a marked improvement from having moved the processing onto the GPU. Using the methods described in section V, the average execution runtime for 45 iterations on the CPU was 8.3044 seconds while the one for the GPU was 1.4280 seconds. These results translate into a performance improvement of approximately 5.8.

From Figure 5, it can be observed that the performance results could have been improved if the first 3 kernels were better optimized. Indeed, these kernels are memory bound and suffer from memory bank conflicts. Due to the project time constraints, these conflicts were not resolved.

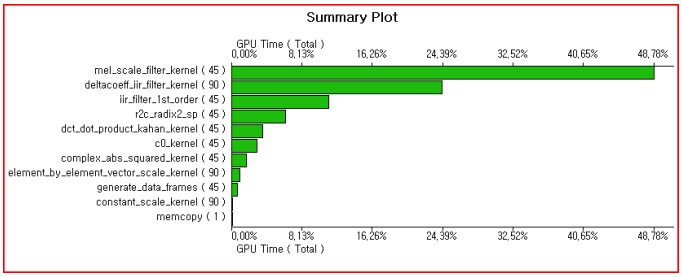

Figure 5: GPU time of each kernel

VII. Conclusion

The current GPU implementation of the isolated-word recognition system demonstrates that it is possible to achieve an increase in performance by moving some of the computationally expensive tasks onto the GPU. Nevertheless, this performance can be further improved by making additional optimizations.

The current implementation of the speech recognizer allocates memory as a contiguous block, which does not guarantee an optimal memory bandwidth usage when kernels are memory bound. In order to address this issue, other CUDA functions (e.g. cudaMalloc2D) should be used to allocate memory so that accesses are aligned. Furthermore, it would be worthwhile to investigate whether the use of the CUBLAS library could yield additional performance improvements since this library has some linear algebra functions which could be applied in the speech recognition program.

References

[1] G. Poli, A. L. M. Levada, J. F. Mari, and J. H. Saito, "Voice Command Recognition with Dynamic Time Warping (DTW) using Graphics Processor Units (GPU) with Compute Unified Device Architecture (CUDA)," *19th International Symposium on Computer Architecture and High Performance Computing*, 2007.

[2] P. Cardinal, P. Dumouchel, G. Boulianne, and M. Comeau, "GPU Accelerated Acoustic Likelihood Computations," *Interspeech 2008*.

[3] J. Chong, Y. Yi, A. Faria, N. R. Satish, and K. Keutzer, "Data-Parallel Large Vocabulary Continuous Speech

Recognition on Graphics Processors," *Proceedings of the 1st Annual Workshop on Emerging Applications and Many Core Architecture (EAMA)*, June 2008.

[4] S. Chu, E. Keogh, D. Hart, and M. Pazzani, "Iterative deepening dynamic time warping for time series," *Proceedings of the 2nd SIAM International Conference on Data Mining*, 2003.

[5] M. Xu, L.–Y. Duan, J. Cai, L.-T. Chia, C.-S. Xu, and Q. Tian, "HMM-Based Audio Keyword Generation," *In Proceedings of the Pacific Conference on Multimedia*, 2004.