OPTIMIZED MFCC FEATURE EXTRACTION ON GPU

Haofeng Kou, Weijia Shang, Ian Lane, Jike Chong
Santa Clara University Carnegie Mellon University

ABSTRACT

In this paper, we update our previous research for Mel-Frequency Cepstral Coefficient (MFCC) feature extraction [1] and describe the optimizations required for improving throughput on the Graphics Processing Units (GPU). We not only demonstrate that the feature extraction process is suitable for GPUs and a substantial reduction in computation time can be obtained by performing feature extraction on these platforms, but also discus about the optimized algorithm. Using one GTX580 GPU our approach is shown to be approximately 97x faster than a sequential CPU implementation, enabling feature extraction to be performed at under 0.01% real-time. This is significantly faster than prior reported results implemented on GPUs, DSPs and FPGAs. Furthermore we demonstrate that multiple MFCC features can be generated for a set of predefined Vocal Tract Length Normalization (VTLN) alpha parameters with little degradation in throughput, along with the optimization for filter bank and reductions.

Index Terms-- Continuous Speech Recognition, MFCC Feature Extraction, Graphics Processing Units, CUDA

1. INTRODUCTION

There have been a number of efforts over the past decades to improve the throughput of MFCC feature extraction. As highly optimized approaches already exist for the CPU [2], research has typically investigated optimizing acoustic feature extraction on a number of specific hardware platforms, including FPGAs [3,4], DSPs [5] and GPUs [6,7,8].The feature extraction approaches typically used in speech recognition are highly parallelizable.

FPGA based acoustic feature extraction has been investigated in prior works, including [3,4]. These approaches all performed significant faster than a CPU implementation and in [4] a 150x relative speedup compared to a CPU/Matlab baseline was obtained. Using this approach MFCC feature extraction was performed at 0.09% real-time. In addition to FPGAs, Digital Signal Processor (DSP) platforms are also well suited for acoustic feature extraction. In [5], real-time MFCC feature extraction was implemented using TMS320C6713 floating point Digital Signal Processor for a simple Support-Vector-Machine SVM-based digit recognition task.

Due to the highly parallel structure GPUs are also well suited for acoustic feature extractions. In comparison to FPGAs and DSPs, GPUs are also much more flexible, a GPU can be used for a variety of tasks, including speech recognition decoding [9] speaker classification [10], and acoustic model [11] and neural-network [12] training. Modern GPU architectures, such as Compute Unified Device Architecture (CUDA) [13], also allow multiple processes to be run synchronously on a single GPU processor. This makes GPUs an ideal platform for high-throughput acoustic feature extraction, as it offers both highly parallel hardware architecture to maximize computational throughput along with the flexibility to perform and switch between complex computational tasks, including speech recognition decoding, and acoustic model training which are difficult to implement on FPGA or DSP platforms. The availability of general-purpose programmable GPU and data parallel programming models [14] has opened up new opportunities for parsing feature extraction at orders of magnitude faster than before. This is further empowered by new algorithms and implementation techniques that focus on parallel scalability [15], which expose the fine-grained concurrency in computation intensive applications and exploits the concurrency on highly parallel many core microprocessors.

A number of prior works [6,7,8] have investigated MFCC feature extraction on GPU platforms. In [6], a speedup of 7x-16x over a CPU implementation was obtained; and in [7], Talakoub and Yi proposed a CUDA-based implementation, which obtained a 5.8x speedup compared to the CPU. They blamed the relatively low throughput obtained on the GPU to open issues with their FFT optimization, and limited optimizations performed within their CUDA implementation, which they have yet to resolve. In [8], Zhang et. al. implemented MFCC feature extraction using CUDA on the Nvidia Fermi architecture. Although all the above approaches obtained significant improvement compared to the CPU, in most cases they used poorly optimized CPU baseline for comparison. For example, in [8], the CPU baseline was approximately 350x slower than the implementation we used in this paper, so even it announces about 348x ratio(T_{cpu}/T_{gpu}) [8], the real speedup is much less than our implementation.

Following up our previous paper [1], we optimize the algorithm by developing a more effective implementation for feature extraction to best leverage the available memory resources and synchronization capabilities. As we know that the computational cost of MFCC extraction takes relatively small portion of entire speech recognition, through our research, we expect to open the opportunity to use GPU for the whole speech recognition.

2. MFCC FEATURE EXTRACTION

MFCC acoustic features are commonly used for feature extraction in modern speech recognition. In this process, the input is the waveform raw date which is divided into frames, of each frame represents 25ms speech signal with 10ms shift between two continuous frames. The processes on parsing different frames are independent and paralleled.

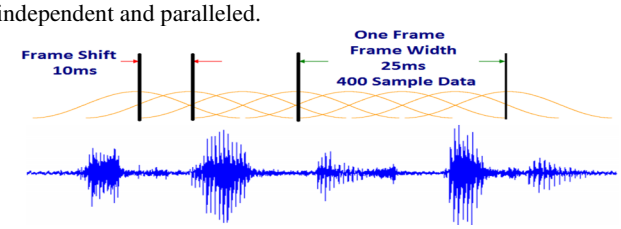

Figure 1: MFCC Feature Extraction Waveform Input

As illustrated in Figure 1 & 2, within a frame, there are 400 sampling waveform data which goes through the following steps of MFCC feature extraction for each frame:

ZMean Frame: Remove DC offset for 400 sample data per frame

$$w(i) = w(i) - \frac{\sum_{i=1}^{FS} w(i)}{FS} \qquad 1 \le i \le FS \qquad (1)$$

w(i): waveform data

FS: frame size -samples per frame

Preemphasis: Boost the energy in high frequencies to enhance the prominence of higher formants for 400 sampling data per frame

$$w(i) = \begin{cases} w(i) - w(i-1) * pec & 2 \le i \le FS \\ w(i) * (1.0 - pec) & i = 1 \end{cases} \quad (2)$$

pec: preEmphasise coefficient

Windowing: Apply Hamming window for sampling data per frame

$$w(i) = w(i) * \left(0.54 - 0.46 \cos\left(\frac{2\pi(i-1)}{FS-1}\right)\right) \quad 1 \le i \le FS \quad (3)$$

Log Raw Energy: Compute frame energies

$$\text{LE} = \log\left(\sum_{i=1}^{FS} w^2(k)\right) \quad (4)$$

Fourier Transform: Extract spectral information for discrete frequency bands using the 400 sampled data per frame to generate the real and imaginary parts for each sampling date

$$e^{j\theta} = cos\,\theta + jsin\theta \quad (5)$$

$$X[k] = \sum_{i=0}^{\text{FS}-1} \text{x[n]} e^{-j\frac{2\pi}{FS}ki} \quad (6)$$

This paper use the GPU-optimized 512x512 FFT code which is generated using the Spiral implementation described in [14]

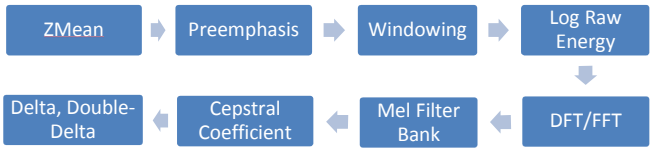

Figure 2: MFCC Feature Extraction Steps

Mel Filter Bank: Apply a Mel-scale filter bank and compute energies in each frequency band. The input contains real and imaginary part of 400 float point data generated by FFT and converts these 800 float point data into 24 filter bank data:

$$mel(f) = 1127\ln(1 + \frac{f}{700}) \quad (7)$$

Step 1:

$$a(i) = \sqrt{re(i-1)^2 + im(i-1)^2} \quad (8.1)$$

Step 2:

$$fb(i, lc(i)) = \begin{cases} fb(lc(i)) + lw(i) * a(i) & lc(i) \ge \text{FN} \\ fb(lc(i)) + a(i) - lw(i) * a(i) & 0 < lc(i) < FN \end{cases}$$
$$\text{LO} \le i \le H \quad (8.2)$$

Step 3:

$$fb(i) = \begin{cases} log(fb(i)) & fb(i) \ge 1.0 \\ log(1) & fb(i) < 1.0 \end{cases} \quad (8.3)$$

fb(i)/lw(i/lc(i)):: filter bank data & channel weights/index

a(i): squart root of filter bank vector

re(i)/im(i): real/imaginary part of filter bank FFT channel

FN: filter bank number

LO/HI: FFT indices of lopass/hipass cut-off

Compute Cepstral Coefficients: Include the following 3 steps:

Calculate 0'th cepstral coefficient

$$\text{C0} = \sqrt{\frac{2}{\text{FN}}}\left(\sum_{i=1}^{FN} \text{fb[i]}\right) \quad (9)$$

Apply DCT to filter bank to make MFCC

$$mfcc(i) = \sqrt{\frac{2}{\text{FN}}} \sum_{j=1}^{FN} \text{fb(j)} * \cos\left[\text{i}\frac{\pi}{\text{FN}}\left(\text{j} - \frac{1}{2}\right)\right]$$
$$0 \le i \le MD - 1 \quad (10)$$

MD: MFCC dim

Weight/Re-scale cepstral coefficients

$$\text{mfcc(i)} = \left(1 + \frac{\text{LT}}{2} * \sin((\text{i} + 1) * \frac{\pi}{\text{LT}})\right) * \text{mfcc(i)}$$
$$0 \le i \le MD - 1 \quad (11)$$

LT: cepstral liftering coefficient

Compute Delta & Acceleration Coefficients

$$delta(t) = \frac{mfcc(t+1) - mfccc(t-1)}{2} \quad (12)$$

$$acc(t) = \frac{delta(t+1) - delta(t-1)}{2} \quad (13)$$

delta(t)/acc(t): delta/ acceleration coefficient

Energy, delta and acceleration features are appended to the cepstral features, resulting in a multi-dimension MFCC feature per frame. In the experimental evaluation performed in this paper a 39 dimension MFCC is generated.

VTLN: Widely used to improve the accuracy of speech recognition systems by reducing the spectral mismatch caused by variations in vocal tract lengths between speakers. In our implementation VTLN is applied as a piece-wise warping of the spectrum when applying the Mel-Filter Bank. A VTLN parameter of α performs a warping of:

$$f_i^a = \begin{cases} af & 0 \le f \le f_r \\ af_r + \left(\frac{\frac{f_s}{2} - af_r}{\frac{f_s}{2} - f_r}\right)(f - f_r), & f_r < f \le \frac{f_s}{2} \end{cases} \quad (14)$$

3. GPU SOFTWARE COMPUTING MODEL

CUDA is the hardware and software architecture that enables NVIDIA GPUs to execute programs written with C and other languages. As illustrated in Figure 3, CUDA program calls parallel kernels which executes in parallel across a set of parallel threads. The programmer or compiler organizes these threads in thread blocks and grids of thread blocks. The GPU instantiates a kernel program on a grid of parallel thread blocks. Each thread within a thread block executes an instance of the kernel, and has a thread ID within its thread block, program counter, registers, per-thread private memory, inputs, and output results.

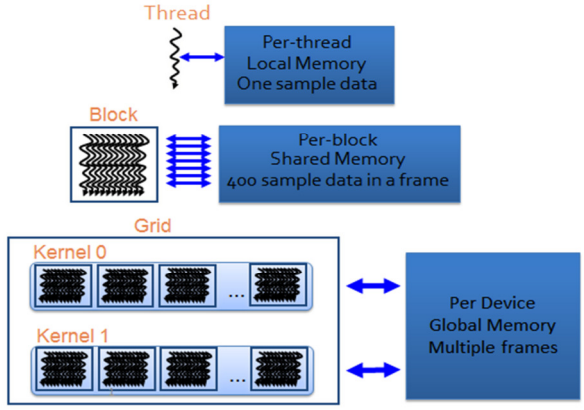

Figure 3: GPU Architecture & Software Computing Model

A thread block is a set of concurrently executing threads that can cooperate among themselves through barrier synchronization and shared memory. A thread block has a block ID within its grid. A grid is an array of thread blocks that execute the same kernel, read inputs from global memory, write results to global memory, and synchronize between dependent kernel calls. In the CUDA parallel programming model, each thread has a per-thread private memory space used for register spills, function calls, and C automatic array variables. Each thread block has a per-Block shared memory space used for inter-thread communication, data sharing, and result sharing in parallel algorithms. Grids of thread blocks share results in Global Memory space after kernel-wide global synchronization.

4. GPU MFCC FEATURE EXTRACTION

Feature extraction is a highly data-parallel operation. Concurrency exists within the processing of both individual frame and across multiple frames. Efficient implementation on a highly parallel GPU involves mapping these levels of concurrency onto available hardware resources while managing data working sets and

synchronization requirements. Figure 3 illustrates the software computing model for MFCC feature extraction on GPU.

4.1 Task Considerations

In our implementation, concurrency within an individual frame is mapped to a thread block, and concurrency across different frames is mapped to different thread blocks.

There exist significant interactions between parallel threads in computing the Log Raw Energy (Equation 4), the FFT (Equation 5 & 6), the Mel Filter Bank (Equation 7 & 8) and the Cepstral Coefficient (Equation 9, 10 & 11). Mapping an individual frame to a thread block allows the efficient interactions between threads to take place. The only data structure that is shared between computations across different frames is the original raw waveform. Since the data structure is read-only, there is little synchronization necessary until the final calculation of the Delta and Double Delta components of the final output.

4.2 Data Considerations

Feature extraction involves a series of computationally inexpensive steps such as Preemphasis (Equation 2), Windowing and Log Raw Energy calculations (Equation 3 & 4). Many of these steps can be easily parallelized. Given the limited availability of the fast shared (scratch pad) memory on a GPU, the data working set must be carefully managed. To do this we buffer the intermediate results in temporary data structures that fit into the shared (scratch pad) memory on the GPU, thus significantly reducing the need to wait for long latency off-chip memory accesses.

4.3 Synchronization Considerations

In the Mel Filter Bank (Equation 7 & 8) and FFT (Equation 5 & 6) step, significant synchronization challenges exist for a highly parallel implementation. We chose to leverage work from the Spiral project [14] for FFT. The Mel Filter Bank (Equation 7 & 8) step collects results from the FFT step and produces 24 Mel channels per frame. This is an instance of the histogram generation problem. Our implementation utilizes the 2 dimension parallel algorithm to shared memory for a fast implementation of this step.

4.4 VTLN Optimizations

For VTLN (Equation 14) we implement the feature extraction steps in two kernels. The first kernel aggregates all steps up to FFT, and the second kernel starts from Mel Filter Bank and completes on final MFCC output. This implementation allows two usage scenarios: when we are calibrating for the VTLN parameter for a speaker, we execute the first kernel once and the second kernel multiple times to compute the MFCC features derived from the intermediate FFT results; when we have determined the VTLN parameter to use, we can execute the first kernel once, and the second kernel once for best performance. Time consumption is reduced from (T1 + T2) * K to T1 + T2 * K.

4.5 CUDA Reduction

As one of the fundamental optimizations of GPU parallel computing, CUDA reduction is a tree-based approach using one or multiple thread blocks to process data in parallel at O(log(n)). Base on the different CUDA reduction approaches [16], we evaluate for each step in MFCC feature extraction and choice the proper one to achieve the best performance:

- Interleaved Addressing: Divergent Branches / Bank Conflicts
- Sequential Addressing
- First Add During Load
- Unroll the Last Warp
- Completely Unrolled
- Multiple Adds/Thread Element Per Thread

4.5 Other Basic Optimizations

Filter bank computation algorithm is different from all other steps, because the index of filter bank array is also a function lc(i) of the index as shown in Equation 7 & 8 and Figure 5. The index of lc(i) is in the range of FFT lopass and hipass cut-off, which means different indexes may lead to the same lc(i) for filter bank. So if it is parallelized at filter bank level, race condition can cause problem for the computation because different i:Index gives the same lc(i) value which is used for fb(lo(i)), for example, lc(150) = lc(160) = 20 makes fb(20) re-entry twice when i:Index is equal to 150 and 160 as shown in Figure 4.

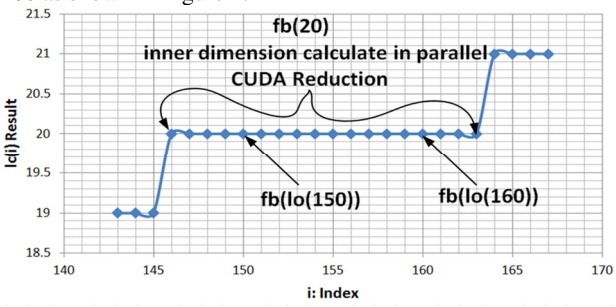

Figure 4: Filter Bank Inner Dimension Calculation

The atomic operation can be used to prevent above race condition, but it will be serialized by GPU which is quite costly. So in order to not only prevent race condition but also not bring in additional delay, we design a 2 dimension parallel algorithm. The idea is to use the 2 dimension computation of which the inner dimension is the filter bank lower channel lo(i) calculation for different i:Index as illustrated in Figure 4; and the outer dimension is in charge of the normal filter bank calculation by passing through all fb(lo(i)) which have the same lo(i) but different i:Index as illustrated in Figure 5. Both dimensions can run in parallel using CUDA based reduction and the loop count is defined based on the filter bank lower channel lo(i) output value.

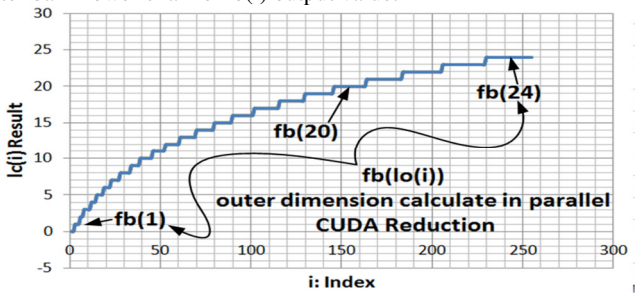

Figure 5: Filter Bank Lower Channel Overview & Filter Bank Outer Dimension Calculation

5. EXPERIMENTAL EVALUATION

We evaluated the effectiveness of our proposed GPU by first comparing the time required to perform one frame of feature extraction to a single thread CPU implementation. The speech corpora used in this evaluation consisted of 28378 utterances totaling 58.4 hours of speech.

5.1 Experimental Setup

For evaluation we used an 3.40GHz Intel Core i7-2600 CPU as the host platform, and an NVIDIA Geforce GTX580 GPU as an accelerator card. The GTX580 is based on the Fermi architecture with 16 cores and 32 lanes per core. There is 64 KB of fast shared memory of which 48 KB can be used as a software scratch pad. For compilation, we used gcc version 4.3.4 and CUDA NVCC 4.1 targeting GPU Compute Capability 2.0. The operating system used was openSUSE 12.1 (x86_64) with Linux Kernel-3.1.9.

Table1: Average time required per frame for 3 major steps

	CPU (µs)	GPU (µs)	Relative Speedup
Preprocessing	14.36	0.088	145x
FFT	89.55	0.870	103x
Feat. Comp.	19.32	0.312	58x
TOTAL	123.23	1.270	97x

5.2 Analysis of Feature Extraction

We analyze the execution times of various steps in MFCC feature extraction in three groups: Pre-processing, FFT, and Feature Computation. The execution times to calculate a frame of features on the CPU and the GPU are compared in Table 1 in units of µs.

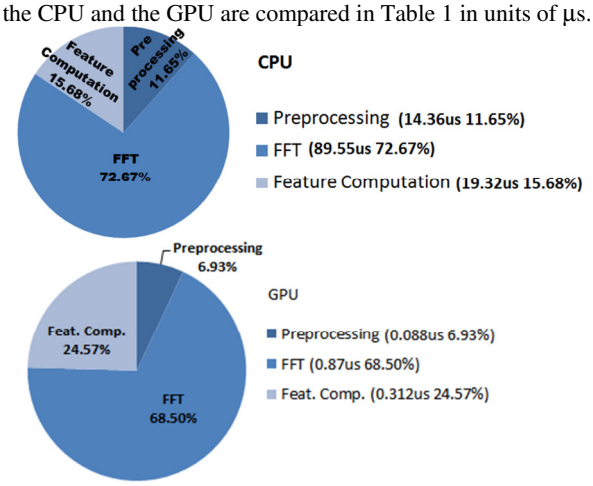

Figure 6: Percentage of time spent on each processing stage

The CPU reference implementation is adapted from Julius 4.2.1 [15]. Our GPU implementation executes 97x faster than the reference CPU version. The preprocessing group produced the most speedup, as the computation there is highly regular and maps well to the GPU. The FFT is a version provided by the CMU Spiral group which produces over 100x speedup compared to the reference implementation. The group of feature computation steps has limited parallelization opportunities and requires significant amount of synchronization in executing a histogram-like function. It achieved 58x speedup compared to the reference implementation.

For MFCC feature extraction, FFT is the dominating step in terms of execution time. Figure 6 show that FFT consumes 72.67% and 68.5% of the feature extraction time for the CPU and GPU implementations respectively.

To compute multiple VTLN (Equation 14), all three groups of steps in the reference implementation must be repeatedly executed. On the GPU, only the Feature Computation steps must be repeatedly executed. The execution times to calculate a frame of features on the CPU and the GPU are compared in Table 2 in units of µs.

The 2000+ relative speedup is achieved by using a less-redundant approach in the feature extraction algorithm on the GPU. It is achieved by reusing the result of the FFT to compute the various versions of MFCC features for different VTLN parameters. The Feature Computation step must be repeated and thus has retained the 50x relative speedup.

For multiple VTLN feature extraction, FFT is no longer the dominating step on the GPU in terms of execution time. As shown in Figure 7, only 11.58% of the total extraction time is spent in FFT on GPU. The majority of the computation time has shifted to the Feature Computation step.

Table2: Average time required per-frame for extracting MFCCs for 21 VTLN factors (0.80 – 1.20 with step size of 0.02)

	CPU (µs)	GPU (µs)	Relative Speedup
Preprocessing	301.51	0.088	3426x
FFT	1880.63	0.870	2161x
Feat. Comp.	405.80	6.552	61x
TOTAL	2587.94	7.51	344x

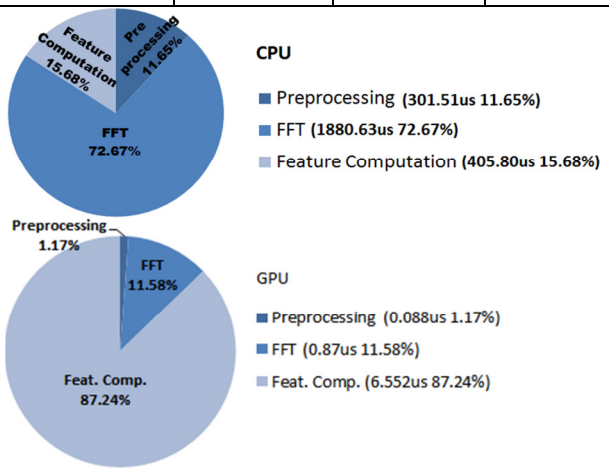

Figure 7: Extraction of MFCCs for 21 VTLN parameters

5.3 Analysis of VTLN Parameter Extraction

Table 3 illustrates the performance of the various implementations we discussed in units of percentage real time factor. One percentage real time factor means that one can extract the features for 100 seconds of audio in 1 second. On a GTX580, we can extract MFCC features with one VTLN parameter (Equation 14) for 2.17 hours of audio in one second. Alternatively, we can extract 21 sets of MFCC features from 23 minutes of audio in one second.

Table 3: Real-time Factor Percentage Table

RTF%	1	11	21
CPU (Baseline)	0.8000%	8.8000%	16.8000%
580 (Baseline)	0.0128%	0.1408%	0.2688%
580 (Multi-VTLN)	0.0128%	0.0413%	0.0729%

6. CONCLUSION

In this paper, we optimized the GPU-based implementation of MFCC feature extraction by introducing the 2 dimension parallel algorithm and custimized CUDA reductions. Using a single Nvidia GTX580 GPU we demonstrated that our proposal approach is 97x faster than a sequential CPU implementation, enabling feature extraction to be performed at under 0.01% of real-time enabling us to extract MFCC features two hours of audio in 1 second. This is significantly faster than prior reported results implemented on GPUs, DSPs and FPGAs. Furthermore, we demonstrated that multiple MFCC features can be generated for sets of predefined VTLN alpha parameters with only a small degradation in throughput. Using the approach described in this paper MFCC features were extracted in 0.07% real-time for 21 VTLN parameters enabling features for all 21 VTLN parameters to be extracted on 2 hours of audio in about 5 seconds.

7. REFERENCES

[1] H.Kou, W.Shang, I.Lane, J.Chong "Efficient MFCC Feature Extraction on Graphics Processing Units" CIWSP` 2013.

[2] M.J. Hunt, "Spectral signal processing for ASR" Proceedings of ASRU, Keystone, Colorado (1999)

[3] R. Hoare, J. Schuster, K. Gupta, "Speech Silicon: A data-driven SoC for Performing Hidden Markov Model based Speech Recognition,". Proc. HPEC 2005, Lincoln Labs, MIT, Boston, MA.

[4] Erik M. Schmidt, Kris West, Youngmoo E. Kim, "Efficient Acoustic Feature Extraction For Music Information Retrieval Using Programmable Gate Arrays" in 10th International Society for Music Information Retrieval Conference, ISMIR 2009

[5] J. Manikandan, B. Venkataramani, K. Girish, H. Karthic, V. Siddharth, "Hardware Implementation of Real-Time Speech Recognition System Using TMS320C6713 DSP" in VLSI Design (VLSI Design), 24th International Conference, 2011

[6] D. Bremer, J. Johnson, H. Jones, Y. Liu, D. May, J. Meredith, and S. Veydia, "Application Kernels on Graphics Processing Units,"in Workshop on HPEC, 2005.

[7] O. Talakoub and A. Yi, "Implementing a Speech Recognition System on a GPU using CUDA", Class report for ECE1742S: Programming Massively Parallel Multiprocessors using CUDA, University of Toronto, retrieved online on 4/2/2010.

[8] L. Zheng,S. Buthpitiya, I. Lane, and J. Chong, "Highly Parallel Computing for Real-Time Large Vocabulary Speech Recognition and Training", LTI Technical Report, Fall 2010

[9] K. You, J. Chong, Y. Yi, E. Gonina, C. Hughes, Y.-K. Chen, W. Sung, and K. Keutzer, "Parallel scalability in speech recognition," IEEE Signal Processing Magazine, vol. 26, no. 6, 2009.

[10] G. Friedland, J. Chong, and A. Janin, "Parallelizing speaker-attributed speech recognition for meeting browsing," in Proc. IEEE International Symposium on Multimedia, Taichung, Taiwan, December 2010.

[11] S. Buthpitiya, I. Lane and J. Chong, "A Parallel Implementation of Viterbi Training for Acoustic Models using Graphics Processing Units", InPAR, 2012

[12] S. Scanzio, S. Cumani, R. Gemello, F. Mana, and P. Laface, "Parallel implementation of artificial neural network training for speech recognition," Pattern Recognition Letters 31: 1302–1309, 2010

[13] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," Queue,vol. 6, pp. 40–53, March2008. http://doi.acm.org/10.1145/1365490.1365500

[14] Franchetti, F., Puschel, M., Voronenko, Y., Chellappa, S., Moura, J.M.F., "Discrete Fourier Transform on Multicore" in Signal Processing Magazine, IEEE , 2009

[15] A. Lee and T. Kawahara, "Recent development of open-source speech recognition engine Julius," In Proceedings of the 1st Asia-Pacific Signal and Information Processing Association Annual Summit

[16] M. Harris, "Optimizing Parallel Reduction in CUDA" Nvidia developer Technology