Available online at www.elixirpublishers.com (Elixir International Journal)

Electrical Engineering

Elixir Elec. Engg. 69 (2014) 23323-23326

ISSN: 2229-712X

Acceleration of speech processing algorithms using reconfigurable hardware

Kamoru Oluwatoyin Kadiri
Department of Electrical/Electronics Engineering Department Federal Polytechnic Offa, Kwara State.

ARTICLE INFO

Article history:
Received: 26 January 2014;
Received in revised form:
30 March 2014;
Accepted: 18 April 2014;

Keywords
Algorithms,
Embedded systems,
DTW- Dynamic Time Warping,
Speech recognizer.

ABSTRACT
Relevant background material about speech recognition is presented, along with a critical review of previous hardware implementations. Accurate real-time speech recognition is not currently possible in the mobile embedded space where the need for natural voice interfaces is clearly important. The continuous nature of speech recognition coupled with an inherently large working set creates significant cache interference with other processes. Hence, real-time recognition is problematic even on high-performance general-purpose platforms. This paper provides a detailed analysis of CMU's latest speech recognizer (Sphinx 3.2.). Several optimizations are then described which expose parallelism and drastically reduce the bandwidth and power requirements for real-time recognition. A special-purpose accelerator for the dominant Gaussian probability phase is developed for a 0:25_ CMOS process which is then analyzed and compared with Sphinx's measured energy and performance on a 0:13_ 2.4 GHz Pentium 4 system. The results show an improvement in power consumption by a factor of 29 at equivalent processing throughput. However after normalizing the process, the special-purpose approach has twice the throughput, and consumes 104 times less energy than the general-purpose processor. The energy-delay product is a better comparison metric due to the inherent design trade-offs between energy consumption and performance. The energy-delay product of the special-purpose approach is 196 times better than the Pentium 4. These results provide strong evidence that real-time large vocabulary speech recognition can be done within a power budget commensurate with embedded processing using today's technology.

Introduction

The idea of being able to talk to a computer, and have it understand one, has been a recurring theme in science fiction for decades. While we are not yet at the stage where computers can comprehend our every word, and act on them, these machines are becoming ever more complex and ubiquitous. But before a computer (or, for that matter, a human being) can attempt to understand speech, it must first convert the audio stream it receives into what that stream actually represents: initially, the basic sounds that make up a language, and ultimately, words. To do that with greater reliability and fidelity, and to be able to cope with different speakers and noisy environments, are the goals of current research in speech recognition. While others concentrate on developing the algorithms and models, there still remains the question of how to implement them. Commercial software packages already exist which can run on a PC — but they are limited by having to operate on a general-purpose processor. In the end, to achieve the maximum processing power, application-specific hardware is the answer. Accordingly, a hardware implementation of a speech recognition system is presented for ubiquitous computing to become both useful and real, the computing embedded in all aspects of our environment must be accessible via natural human interfaces. Future embedded environments need to at least support interfaces such as speech (this paper's focus), visual feature recognition, and gesture recognition. A viable speech recognizer needs to be speaker independent, accurate, cover a large vocabulary, handle continuous speech, and have implementations amenable to mobile as well as tethered computing platforms. Current systems fall short of these goals primarily in the accuracy, real time, and power requirements.

Review of related literature

Speech recognition is a computationally demanding task, especially the decoding part, which converts pre-processed speech data into words or sub-word units, and which incorporates Viterbi decoding and Gaussian distribution calculations. Most speech recognition research has targeted recognition accuracy. Performance issues have been secondary and power efficiency has largely been ignored. Ravishankar improved Sphinx performance by reducing accuracy and subsequently recovering it in a less computationally active phase and developed a multi-processor version of an older version of Sphinx. However, details of these works are currently unavailable. Agaram provided a detailed analysis of Sphinx 2 and compared this analysis with SPEC benchmarks. Pihl designed a 0:8_ custom coprocessor to accelerate Gaussian probability generation for an HMM based recognizer. However, Pihl's work proposed a specialized arithmetic format rather than the IEEE 754 compatible version described here. Furthermore, the number of Gaussian components need to be processed per second has escalated from 40,000 in the case of Pihl's coprocessor to 4.9 million for our accelerator during the last 7 years and this trend is likely to continue as the search for increased accuracy proceeds.

Pihl's work did not address scalability which is a central theme for this research. Tong showed an example of reduced precision digit serial multiplication for Sphinx. Anatharaman showed a custom multiprocessor architecture for improving the Viterbi beam search component of a predecessor of Sphinx. Application acceleration using custom coprocessors has been in use for decades, However, current researchers are exploiting this theme for reducing power consumption. Piperench is one

Tele:
E-mail addresses: Kadiritoyin2007@yahoo.com

approach which exploits virtualized hardware, and run-time reconfiguration. Pleiades is a reconfigurable DSP architecture that uses half the power of an Intel Strong ARM for FFT calculation.

Aims and objectives

The aim of this research is to design and implement a speech recognition system, with the decoding stage implemented in hardware, in order to:

• assess the suitability of so doing for the various parts of the recognition algorithm;

• compare the processing speed of hardware and software implementations, in order to ascertain the possible speedup;

• determine the requirements inherent in applying hardware (field-programmable gate array (FPGA).to speech recognition.

Justification of research

This work is based on CMU's Sphinx 3 system. Sphinx 3 uses a continuous model that is more accurate than the previous semi-continuous Sphinx 2 system but requires significantly more compute power. Sphinx 3 runs at 1.8x slower than real time on a 1.7 GHz AMD Athlon. Performance is hardly the problem since improvement rates predicted by Moore's Law assures that real time performance will be available soon. A much more important problem is that the real time main memory bandwidth requirement of Sphinx 3 is 800 MB/sec. Our 400 MHz Strong ARM development system has a peak bandwidth capability of only 64 MB/sec and this bandwidth costs 0.47watts of power. A reasonable approximation is that power varies with main memory bandwidth for Sphinx 3 indicating that this program is at least an order of magnitude too slow and consumes an order of magnitude too much power for embedded applications. This provides significant motivation to investigate an alternate approach.

Contributions of the Coprocessor Architecture

The main contributions of our coprocessor architecture are energy savings, server scalability and bandwidth savings.

Energy Savings

Without considering the power consumed by main memory, the GAU accelerator consumed 1.8 watts while the Pentium 4 consumed 52.3 watts during Mahanalobis distance calculation, representing an improvement of 29 fold.

Scalability

In addition to having energy advantages, our design is also scalable. The main limitation is our in-order processor with its simple blocking cache mode. The Final Sigma stage enables the design to scale even with blocking caches due to the removal of destructive interference between the cache and the DMA engine

Bandwidth Savings

Because of our blocking optimization (GAU OPT), we need to process the data only 10 times per second with a peak bandwidth of 180 MB/s which can be further reduced by applying the sub-vector quantization (non-feedback) heuristics in Sphinx. Not only does our design bring the bandwidth requirements to limits possible on embedded systems, it also drastically improves the power consumption

Methodology

A typical speech recognition system consists of three stages:

First Stage

The pre-processing stage which takes a speech waveform as its input, and extracts from it feature vectors or observations which represent the information required to perform recognition. Automatic speech recognition systems make use of the modulation applied by the vocal tract (throat, tongue, teeth, lips and nasal cavity); the excitation produced by the larynx is not used, even though humans infer much information from it. Converting a speech waveform into a form suitable for processing by the decoder requires several stages. A typical such process is as follows:

1. The waveform is sent through a low pass filter, typically 4 to 8 kHz. As is evidenced by the bandwidth of the telephone system being around 4 kHz, this is sufficient for comprehension.
2. The resulting waveform is sampled. Sampling theory requires a sampling rate of double the maximum frequency (so 8 to 16 kHz as appropriate).
3. The data undergoes frequency analysis using a discrete Fourier transform. This produces information about the frequency within each analysis window, which is typically 20ms wide, with each one overlapping its neighbour by 10ms.
4. Human hearing is not particularly sensitive to phase, so this information is removed by taking the modulus of the complex frequency data.
5. Loudness is perceived by humans on a log scale, rather than a linear one, so the log of the power is computed for the frequency data.
6. Frequency is also perceived on a non-linear scale. In particular we discriminate better between low frequency sounds than high frequency ones, and so the mel-scale is used to compensate for this. A filter bank analysis is performed, whereby the frequency magnitudes are grouped into a number of bins, with the bins spaced out according to the mel scale so as to take account of our non-linear perception. Twelve of such bins are typical.
7. In order to make recognition calculations less complex (specifically, to ensure that the covariance matrix is diagonal), it is required that the mel-scale filterbank components be uncorrelated, which is not normally the case. In order to achieve this, a discrete cosine transform is effected, as a more computationally efficient approximation to principal component analysis, in order to a produce a set of mel-frequency cepstral coefficients (MFCC).
8. An additional parameter can be added in the form of an energy term, computed as the log of the signal energy.
9. Finally, further information about the "shape" of the speech data can be obtained by taking the first and second derivatives of the cepstral coefficients. Hence, starting with twelve bins, adding an energy value, and then taking the derivatives, we end up with a 39-dimensional vector.

An alternative to mel filterbank analysis is linear prediction, where the vocal tract is modelled by a transfer function, and the filter coefficients are calculated from the data in order to minimise the prediction error.

Second Stage

The recognition or decoding, which is performed using a set of statistical models, known as hidden Markov models (HMMs). At their simplest, the HMMs represent monophones, i.e. the basic distinct sounds of a particular language, of which English has around 50. However, when people speak, these sounds are affected by those uttered immediately before and after them. In order to model this effect, a larger number of models, now representing pairs and triplets of monophones (biphones and triphones), can be used, leading to improved recognition ability. In addition, a language model can be used, which contains further information as to the probability of one recognition unit (monophone or biphone/triphone, as appropriate) following another.

For small- to medium-sized vocabularies, the word and language models are compiled into a single, integrated model. Recognition is performed using the Viterbi algorithm to find the

route through this model which best explains the data. For large vocabulary systems, this approach is not viable due to the large size of the search space, and so methods of restricting its size are required. Besides the standard practice of pruning the least likely paths, this can be achieved by incorporating other information, such as data based on language usage or the formation of speech, by using multiple passes, or by heuristic methods such as stack decoding.

Third Stage

In the third stage, word-level acoustic models are formed by concatenating the recognition units according to a pronunciation dictionary. The word models are then combined with a language model, which constrains the recognizer to recognize only valid word sequences.

The first and third stages can be performed efficiently in software (though some of the pre-processing may be better suited to a DSP). The decoding and associated observation probability calculations, however, place a particularly high load on the processor, and so it is these parts of the system that have been the subject of a number of implementations in hardware, often using custom-built chips. However, with ever more powerful programmable logic devices (PLDs) being available, such chips appear to offer an attractive alternative.

Data Analysis

We performed a few tests to gain a sense of the error rates of our system. Two sets of tests were performed; one where each DTW trained stored a unique word, and one where sets of three DTWs were trained per word. This initial set of tests examined the performance of the algorithm itself, while the latter tests examined more closely the performance of the system as we planned to use it in gaming applications.

To complement our initial Matlab tests between the words \taco" and _sh" we examined our system's recognition of these two words. Each word was uttered 10 times, and the number of matches for each word was recorded. The vertical chart of the following graph describes the two words trained into the system, while the horizontal axis describes the word being used to test.

The final algorithmic test we attempted, and arguably the most difficult, attempted to distinguish the words \alpha," \bravo," \charlie," and \delta" from each other. In this case, each word was uttered 5 times, yielding the results depicted below. Trained Word alpha bravo charlie delta

This test demonstrated some significant problem areas in our algorithm. First, our algorithm had difficulty detecting the utterance of a fricative, due to the high-frequency low-magnitude nature of fricatives. This accounts for the lack of successful matches with the word \charlie," since that test point starts with a fricative-like sound. This test also compared two words that sounded very similar: \alpha" and \delta." Unsurprisingly, our algorithm had difficulty distinguishing such similar words from each other, matching each word approximately equally when the uttered word was \delta." This may be due to the soft or quiet nature of the initial consonant, which may not have been detected by our system in many cases. To combat high error rates when used in practical applications, we decided to operate our system with 3 DTWs trained on 3 instances of each word. We conducted similar error-rate tests with this modified configuration, using words of more pertinent interest to the system's intended use. In both of the tests described below, each word was uttered 10 times in testing, and a word match was counted if any of the DTWs associated with some words successfully matched the input. First, we examined the error rates when given the words. With this modification to the application of our algorithm, we were able to achieve substantially better error rates on words we expected our system to handle. This result demonstrated promise for our system to effectively recognize words in a gaming setting.

Discussion

A simplistic view of the high-level organization of Sphinx 3 is shown in Figure 1. Rectangles represent algorithmic phases and rounded boxes represent databases. The numbers in parenthesis are the approximate on-disk size of the databases before they are loaded into memory and possibly expanded. Sphinx has 3 major logical phases: front-end signal processing which transforms raw signal data into feature vectors; acoustic modeling which converts feature vectors into a series of phonemes; and a language model based search that transforms phoneme sequences into sequences of words. The process inherently considers multiple probable candidate phoneme and word sequences simultaneously. The final choice is made based on both phoneme and word context. We focus on analyzing the dominant processing component of the acoustic and search phases in this paper. The front end will hereafter be referred to as FE. The dominant computation done during acoustic model evaluation is Gaussian probability estimation. Hence, the figure and the rest of this paper refer to this algorithm known as GAU.

The key component of the search phase is Hidden Markov Model evaluation. So, we refer to it as HMM. A more accurate and detailed view is that Sphinx models language using hidden Markov models where the probability of observing a feature vector while in a particular state is assumed to follow a Gaussian distribution. GAU precomputes Gaussian probabilities for sub-phonetic HMM states (senones). The output of the GAU phase is used during acoustic model evaluation and represents the probability of observing a feature vector in an HMM state. The Gaussian probability is computed as the weighted sum of the Mahanalobis distance of the feature from a set of references used while training the recognizer. The Mahanalobis distance is a statistically significant distance squared metric between two vectors. Given a feature vector Feat and the pair of vectors (M;V) (hereafter called a component) which represent the mean and variance from a reference, GAU spends most of its time computing the quantity:

The Gaussian reference table contains 49,152 components for the HUB4 speech model we use. Each component consists of an instance of a mean vector and a variance vector. Sphinx uses feedback from the HMM phase to minimize the number of components GAU needs to evaluate. In the worst case, every single component needs to be evaluated for every single frame. A real time recognizer should have the ability to perform 4.9 million component evaluations per second. In practice, the feedback heuristic manages to reduce this number to well under 50%. The Viterbi search algorithm for HMMs is multiplication intensive, but Sphinx as well as many other speech recognizers convert it to an integer addition problem by using _xed point arithmetic in a logarithmic domain. FE and GAU are the only oating-point intensive components of Sphinx.

The Sphinx 3 code spends less than 1% of its time on front end processing, 57.5% of the time on the Gaussian phase and 41.5% on the HMM phase. While our work has addressed the entire application, the work reported here addresses the optimization and implementation of the dominant Gaussian phase. The contributions include an analysis of the Sphinx 3 system, an algorithmic modification which exposes additional parallelism at the cost of increased work, an optimization which drastically reduces bandwidth requirements, and a special-purpose coprocessor architecture which improves the performance of Sphinx 3 while simultaneously reducing the

energy requirements to the point where real-time, speaker-independent speech recognition is viable on embedded systems in today's technology.

Findings

Sphinx has a lengthy startup phase and extremely large data structures which could cause high TLB miss rates on embedded platforms with limited TLB reach. To avoid performance characteristics being aliased by startup cost and the TLB miss rate, Sphinx 3.2 was modified to support check pointing and fast restart. For embedded platforms, the check-pointed data structures may be moved to ROM in a physically mapped segment similar to kseg0 in MIPS processors. Results in this paper are based on this low startup cost version of Sphinx referred to as original. Previous studies have not characterized the 3 phases separately. To capture the phase characteristics and separate optimizations for embedded architectures, we developed a phased" version of Sphinx 3. In phased version, each of the FE, GAU and HMM phases can be run independently with input and output data redirected to intermediate files. In the rest of this paper FE, GAU, HMM refers to the corresponding phase run in isolation while phased refers to all three chained sequentially with no feedback. In Phased, FE and HMM are identical to original, while GAU work is increased by the lack of dynamic feedback from HMM.

It appeared likely that a multi-GHz processor might be required to operate Sphinx in real time. Parameters like L1 cache hit time, memory access time, oating-point latencies etc were measured on a 1.7GHz AMD Athlon processor using the lm bench hardware performance analysis benchmark.

In the phased version, we found that approximately 0.74%, 55.5% and 41.3% of time was spent in FE, GAU and HMM respectively. Since FE is such a small component of the execution time, we ignore it in the rest of this study and concentrate on the analysis of the GAU and HMM phases.

Conclusion

We implemented an isolated word speech recognition system in hardware. This system processes raw audio data using Mel scale values and matches words using the dynamic time warping algorithm. We have successfully demonstrated this systems capability to distinguish many words from each other, through the use of our algorithm and through the use of multiple DTW modules trained on the same word.

Our system works well at distinguishing dissimilar words from each other. However, there are a number of points in which recognition does not work as well. Words that begin or end with fricatives often have difficulties on account of the high-frequency nature of fricative sounds failing to generate sufficient energy for the system to detect. In a future implementation we may attempt to combat this effect with a pre-emphasis filter that amplified higher frequencies, although initial testing with such a filter tended to increase these higher frequency values too much, hurting the system's ability to match words.

Very short words tend to match much more frequently, since the dynamic time warping algorithm will tend to find a good matching with such short inputs. We attempted to combat this effect by adding a small punishment value for matching words of dramatically different lengths, but properly calibrating this punishment value proved tricky on initial implementation.

In general, we found that our algorithm was effective at matching words and identifying the correct word from the best distance value. However, reasonable threshold values were difficult to determine, and different length words often needed different threshold values. Dynamic threshold values based on word lengths may be useful to implement to improve our system's effectiveness in the future.

Recommendation

Though the Gaussian estimator was designed for Sphinx 3 and the MIPS-like embedded processor, the results are widely applicable to other architectures and recognizers. There are several levels at which this system may be integrated into a speech recognition task pipeline similar to Phased. For example, an intelligent microphone may be created by using a simple low-power DSP to handle the A/D conversion and FE phase and then a GAU coprocessor attached to the DSP may be used for probability estimation. The probability estimates can then be sent to a high-end processor or custom accelerator that does language model computation thereby hiding more than 50% of the compute effort required for speech recognition. On desktop systems, the Gaussian accelerator may be part of a sound card or the Gaussian accelerator may be directly attached to the main processor. On commercial voice servers, the Gaussian estimator may be directly built into the line cards that interface to the telephone network thereby freeing up server resources for language model and application processing.

References

Keogh, Eamonn and Michael Pazzani. Derivative Dynamic Time Warping. November 2008.

K. Agaram, S. W. Keckler, and D. Burger. A characterization of speech recognition on modern computer systems. In Proceedings of the 4th IEEE Workshop on Workload Characterization, Dec. 2001.

J. G. F. David Pallett and M. A. Przybocki. 1996 preliminary broadcast news benchmark tests. In Proceedings of the 1997 DARPA Speech Recognition Workshop, Feb. 1997.

X. Huang, F. Alleva, H.-W. Hon, M.-Y. Hwang, K.-F. Lee, and R. Rosenfeld. The SPHINX-II speech recognition system: an overview. Computer Speech and Language, 7(2):137{148, 1993.

C. Lai, S.-L. Lu, and Q. Zhao. Performance analysis of speech recognition software. In Proceedings of the Fifth Workshop on Computer Architecture Evaluation using Commercial Workloads, Feb. 2002.

R. Mosur. Efficient Algorithms for Speech Recognition. PhD thesis, Carnegie Mellon University, May 1996. CMU-CS-96-143.

M. Seltzer. Sphinx iii signal processing front end specification. http://perso.enst.fr/~sirocco/ May 2002.