Design and Evaluation of an Ultra Low-power Human-quality Speech Recognition System

DENNIS PINTO, JOSE-MARÍA ARNAU, and ANTONIO GONZÁLEZ,
Universitat Politècnica de Catalunya

Automatic Speech Recognition (ASR) has experienced a dramatic evolution since pioneer development of Bell Lab's single-digit recognizer more than 50 years ago. Current ASR systems have taken advantage of the tremendous improvements in AI during the past decade by incorporating Deep Neural Networks into the system and pushing their accuracy to levels comparable to that of humans. This article describes and characterizes a representative ASR system with state-of-the-art accuracy and proposes a hardware platform capable of decoding speech in real-time with a power dissipation close to 1 Watt. The software is based on the so-called hybrid approach with a vocabulary of 200K words and RNN-based language model re-scoring, whereas the hardware consists of a commercially available low-power processor along with two accelerators used for the most compute-intensive tasks. The article shows that high performance can be obtained with very low power, enabling the deployment of these systems in extremely power-constrained environments such as mobile and IoT devices.

CCS Concepts: • **Computer systems organization** → **Other architectures**; *Embedded hardware*; *Neural networks*; • **Computing methodologies** → **Speech recognition**;

Additional Key Words and Phrases: Hardware accelerators, low-power hardware

ACM Reference format:

Dennis Pinto, Jose-maria Arnau, and Antonio González. 2020. Design and Evaluation of an Ultra Low-power Human-quality Speech Recognition System. *ACM Trans. Archit. Code Optim.* 17, 4, Article 41 (November 2020), 19 pages.

<https://doi.org/10.1145/3425604>

1 INTRODUCTION

Computers are getting *smarter* every day. The recent explosion on technologies related to Artificial Intelligence (AI) is transforming the way we interact with machines. The evolution of AI is allowing computers to understand what they see and what they hear, and act accordingly. At some point in the future, keyboards will be reserved for a subset of specialized tasks, and the most common scenarios will be handled by speech interfaces.

This work has been supported by the CoCoUnit ERC Advanced Grant of the EU's Horizon 2020 program (grant No. 833057), the Spanish State Research Agency under grant TIN2016-75344-R (AEI/FEDER, EU), the ICREA Academia program, and the Spanish MICINN Ministry under grant BES-2017-080605.

Authors' addresses: D. Pinto, J.-M. Arnau, and A. González, Universitat Politècnica de Catalunya, C/ Jordi Girona, 1-3, Barcelona, Spain, 08034; emails: {dpinto, jarnau, antonio}@ac.upc.edu.

This work is licensed under a Creative Commons Attribution International 4.0 License.

© 2020 Copyright held by the owner/author(s).

1544-3566/2020/11-ART41

<https://doi.org/10.1145/3425604>

The introduction of graphical user interfaces represented a revolution that allowed the general public to interact with computers, expanding greatly the number of use cases for these machines, to the point that today, almost everybody knows how to use a computer and even owns a couple of them. Written commands are mainly reserved for experts. Accordingly, the next revolution in human-machine interaction will probably be driven by speech interfaces. It is not difficult to foresee the huge impact that a reliable speech interface would have in the way we interact with technology. Using a computer will require no training at all. Even the concept of *using a computer* will become blurred. To reach that point, there are several problems to solve. How do we understand sentences?; how do we correlate concepts?; how do we choose a valid answer to a question?; or the question we study in this work, which is *how is speech encoded in human voice signals*? That problem is known as *Automatic Speech Recognition* (ASR), and researchers have been trying to solve it for more than half a century.

Probably the contribution that influenced most the evolution of ASR was the introduction of Hidden Markov Models (HMMs) [28] in the field of speech. They are used as generative models of “sounds,” in which sounds are modeled by states in a graph, with a probability of moving from one state to another, and each state having a probability distribution of generating each possible signal. Whenever a sequence of sounds—observations—has to be decoded, the sequence of states with a higher probability of generating such sequence of observations is chosen, and the sounds that they represent are predicted to be the transcription of the utterance. Up until recently, the probabilities of observations in HMMs were modeled by Gaussian Mixture Models (GMMs). However, GMMs imply the acceptance of several assumptions, such as the approximation that signal frames are independent [2]. The most recent breakthrough in ASR systems was the substitution of GMMs by Deep Neural Networks (DNNs), to conceive what is known as the hybrid HMM-DNN system, which largely improved the level of accuracy. In Reference [9], several models, based on both DNN and GMMs, are tested on different tasks, to reach the conclusion that systems including DNNs can easily reach GMM performance with less training data, if appropriate training techniques are used, and outperform them when more techniques are applied, such as using more training data or fine-tuning training parameters.

To improve the performance of the system under more general conditions, some kind of speaker adaptation is usually included. Speaker adaptation refers to a class of techniques aimed at improving the recognition for multi-speaker systems. Popular techniques, such as Vocal Tract Length Normalization [34] or Maximum Likelihood Linear Regression [6], are extensively discussed in the literature. The focus in all of them is to transform the signal to remove noise introduced by the specific characteristics of the speaker’s voice. A more recent approach [12] consists on extracting an i-vector [5] from the signal, and append it to the Mel Frequency Cepstral Coefficients (MFCC) feature vector to feed the DNN-based acoustic model.

In addition to the HMMs, other models are used to restrict the combinations to those existing in the particular language and provide additional information about the language. By employing a WFST framework [21], HMMs, lexicon and language models can be merged in a natural and efficient way, resulting in a single graph containing the information from all the sources.

For large vocabulary systems (hundreds of thousands or even millions of words), the resulting graph requires hundreds of MBs or even several GBs, which motivated the research in techniques to reduce the memory requirements, such as language model re-scoring [16, 22] and on-the-fly composition [10, 11].

The current performance of these systems opens the door for very interesting applications, as we see in our everyday. Nowadays, by using our voice, we can open an application or call some of our contacts in our mobile phone, set a route for the GPS to follow, or write a document, to name a few.

AI companies are interested in deploying their speech recognition services and providing APIs to allow developers to integrate those services with their applications. However, the current commercial approach is to record an audio file and send it to the servers of a company providing an ASR service to receive the transcribed text. This approach is limited because of several reasons:

- (1) The requirement for a network connection limits the use of ASR in environments where that is not always an option.
- (2) As the network components are among the most power demanding on mobile devices [25], relying on network services is not the most energy-efficient solution.
- (3) To make an efficient use of the network, speech frames are not decoded one-by-one; instead, an entire utterance is captured and sent. This links the latency of the decoding to the latency of the network and does not allow to perform online decoding, in which the utterance is decoded while it is being produced.
- (4) Sending recorded audio to a server raises some security concerns related to who is receiving that data, how it is being used, and what measures they are taking to protect the data.

We believe that these alone are strong reasons to pursue the development of solutions that perform ASR in the local device rather than on the cloud. Local (on-edge) ASR, however, comes with some challenges. To obtain high recognition accuracy, modern ASR systems include large neural networks (several tens or hundreds of millions of parameters) as Acoustic Models, and perform a Beam Search over graphs containing millions of nodes and edges. This is often complemented with a Language Model Re-scoring pass, which consists on the evaluation of another large Recurrent Neural Network. This results in a high number of computations per second to achieve real-time, as well as high memory footprint and memory bandwidth requirements.

In this article, we evaluate and characterize an ASR system that delivers state-of-the-art recognition accuracy, even outperforming humans in some benchmarks. This system is built by using Kaldi [26], a popular ASR toolkit written in C++/CUDA. By running it on a modern mobile System-on-Chip (SoC) that features an ARM CPU and a low-power GPU, we identify the main performance and energy bottlenecks in the overall ASR pipeline. To alleviate those bottlenecks, we employ two specialized hardware accelerators. These are based on accelerators proposed in the literature, which we adapted by reducing the size of some of its components. Our hardware-accelerated ASR system delivers state-of-the-art accuracy and real-time execution with a power dissipation of around 1 Watt, making it suitable for extremely power-constrained environments such as mobile and IoT devices. By making use of specialized accelerators, the proposed platform reduces the energy consumption by 4.3x, with 4.5x speedup, compared to an *Nvidia Jetson TX1*, a representative mobile SoC.

This article focuses on high-performance and low-power ASR. Its main contributions are the following:

- We characterize a state-of-the-art 200K word ASR system on a modern mobile SoC. We identify the TDNN (acoustic model), the Viterbi beam search, and the RNN-based language model as the main performance and energy bottlenecks. Furthermore, we show that real-time ASR is not achieved for some utterances, even when using the mobile GPU.
- We propose a novel hardware platform that combines a low-power CPU and several hardware accelerators to alleviate the main bottlenecks.
- We evaluate our hardware-accelerated mobile ASR platform using Librispeech dataset. Our system achieves real-time performance for all the utterances, dissipates around 1 Watt of power, and delivers state-of-the-art speech recognition accuracy.

The rest of the article is organized as follows: Section 2 introduces the different components of the ASR system, Section 3 is a detailed description of the hardware accelerators included in our ASR solution, Section 4 is an analysis of our solution as compared to other solutions based on CPU or GPU, and Sections 5 and 6 discuss the related work and conclusions.

2 STATE-OF-THE-ART ASR

There are two main approaches to Automatic Speech Recognition. The prevalent and most mature approach is the so-called *Hybrid System* [2, 8, 9] and is the evolution of the traditional system based on HMMs [29]. The hybrid system is still based on HMMs, but it incorporates a Deep Neural Network to classify the input signal's frames, instead of using a Gaussian Mixture Model, as the earlier ASR systems used to do. An important concern is the high complexity of the system and the high level of expertise required to build and train it. This complexity motivated the so-called End-to-End (E2E) ASR [7, 17], which tries to remove complexity from the hybrid system by training a Neural Network to do the whole decoding process, and thus removing the need for the rest of the components.

Recent work on E2E ASR has reported promising results, even reaching state-of-the-art recognition accuracy (a more detailed comparison is provided in Section 4.1). However, E2E systems are essentially very similar to hybrid systems; they both perform DNN inference over audio features (usually MFCC) to compute a probability distribution over a set of tokens, which is used to perform a Beam Search over a graph containing language information. Both systems can also benefit from a Language Model re-scoring pass. This means that, although this work is focused on a *Hybrid system*, the proposed ideas are not limited to *Hybrid systems* and can be easily applied for E2E systems as well.

The system used for this study is built using the *Kaldi ASR toolkit* [26]. Kaldi is an open-source toolkit developed by a group of researchers in Speech Recognition to support the research and development in the area. The library includes most of the functionality commonly used for ASR, allowing researchers/developers to build any kind of system by putting together existing components or integrating new ones. ASR systems are built in Kaldi through *recipes*. Recipes are nothing more than scripts that prepare the training data sets and run the binaries of different components (DNN, WFST composition...), passing the resulting data from one stage to the next. Kaldi provides multiple recipes specifically tailored to different Speech Corpus and recognition tasks. For this study, we employ a recipe for the Librispeech [23] corpus that trains an ASR system that achieves state-of-the-art performance. From now on, this system will be referred to as the *Kaldi system*.

The first component in the *Hybrid system* is the *Feature Extraction*. This component splits the audio signal in overlapping frames of 25 ms of speech. Next, it computes a vector of *features* to encode each frame. The objective of this representation is to expose the information that is relevant for the system in a compact manner.

The next step is the evaluation of the Acoustic Model. This model is a DNN that classifies the speech frames by computing for every frame the probability of it representing each of the possible sound units (referred to as sub-phonemes, or senones¹) included in the model. These probabilities are used by the next component, the *Decoding*, which consists of a search of the best path in a graph that includes all the possible sequences for the transcription that is known as the *Decoding Graph*. This graph is generated by combining a language model with the acoustic scores. The language model is represented through a *Weighted Finite State Transducer* (WFST), which represents all the allowed transitions among sub-phonemes and their associated probability.

¹These sound units, sub-phoneme units, or senones, are normally defined as context-dependant phonemes, being the most popular the tri-phoneme. A tri-phoneme is composed of a central phoneme p plus left and right phonemes $p_l - p - p_r$.

In the *Kaldi* system, instead of extracting only the very best path from the decoding graph, a set of best paths along with their computed probabilities is generated and fed to the next step, the *Language Model Re-scoring*. This step modifies their probabilities before extracting the definitive best path. The obtained sequence represents the final transcription. This step of Language Model re-scoring is done by using the *pruned composition algorithm* described in Reference [35], which employs a heuristic of the best complete path to decide which partial path to continue re-scoring, and stopping at a threshold between the current and the predicted best path. This heuristic allows for a more efficient execution and better accuracy.

The described process of decoding, including a Language Model re-scoring, is usually called *2-step decoding*, as opposed to *1-step decoding*, which extracts the best path directly from the decoding graph.

The following subsections provide a deeper description of each of the components previously mentioned and the specific details for the Kaldi system.

2.1 Feature Extraction

The input signal, received as a stream of amplitude values at a sampling rate of 16kHz , is first split into overlapping frames of 25ms shifted by 10ms each, which are then transformed into *features*. The features that represent each frame of the signal for the Kaldi system are a combination of two vectors; a vector of *Mel Frequency Cepstral Coefficients*, or *MFCC*, with 40 components per frame, and an *i-vector* of dimension 100.

The MFCC vector is a representation of the signal frame, which emulates the information captured by our ears. The process of computing an MFCC vector includes performing a Fourier analysis on the signal frame to decompose it into frequencies, remove those lying outside the common audible spectrum, and grouping and weighting the rest, according to the *Mel Scale*. A more in-depth description of MFCC and a comparison of different implementations is provided in Reference [40].

To provide more information to the Acoustic Model, an *i-vector* [5] is chained to the MFCC. It is a vector that represents some properties of the speaker's voice that have a good degree of independence from what he or she is saying. They are commonly used for speaker verification, but *i-vector* has also been proven to improve the recognition accuracy in ASR [12].

2.2 Acoustic Model

The Acoustic Model consists of a DNN trained to compute a set of probabilities for each input frame. More specifically, it receives a Feature vector and computes the probability that it corresponds to each of the states in the decoding graph (each state corresponds to a sub-phoneme unit). The specific DNN included as acoustic model for the Kaldi system is a *Time Delay Neural Network* (TDNN) [33]. Although the use of TDNNs for speech recognition was proposed long ago, they have been proven recently to improve the accuracy of state-of-the-art systems more efficiently than other approaches [24], such as using recurrent networks or introducing long contexts to a DNN.

The main idea behind a TDNN is to have a network that is suited to recognize sequences but does not have the overhead of Recurrent Neural Networks (RNNs). It is a feed-forward network whose layers not only receive their input from the previous layer of the current frame, but also from past and future frames. Figure 1 shows an example of the dependencies between layers on a simple TDNN network. In this case, every layer depends on the output from the previous layer in the time-steps $t - 1$, t , and $t + 1$.

Using this network instead of an RNN allows for exploiting more parallelism by removing some dependencies between time-steps. However, as the layers still depend on results from both past and future frames, there are some data dependency constraints that have to be taken into account.

Fig. 1. Example of dependency between layers on a TDNN network.

2.3 Decoding

The Decoding step is the central component of the ASR pipeline. It is a search of the transcription with the highest probability of matching the input signal, according to the models included in the system. More technically, it consists of a Viterbi Beam search over a WFST graph.

The WFST [20], which stands for *Weighted Finite State Transducer*, is a type of weighted graph, over which there are defined operations that allow to efficiently combine information from several models into the same graph. The use of a WFST makes it possible to merge together the HMM, a lexicon, and a language model into the same graph.

Each arc in a WFST graph contains a weight (a.k.a. cost) and two labels: an input label from some label set, or *Alphabet*, A , and an output label from another label set, B . With this graph, a sequence of labels α from A can be converted to a sequence of labels β from B by looking at all the paths in the graph for which the sequence of input labels is equal to α . Among all the possible paths, the one with the lowest cost is chosen, and the sequence of output labels is β .

For the case of a decoding graph, the alphabet A is a set of senones, and the alphabet B is a set of words. However, as mentioned in the previous section, a sub-graph from the decoding WFST is extracted, instead of the single best path, to do language model re-scoring. This sub-graph, called *lattice*, is a WFSA (*Weighted Finite State Acceptor*), which is a WFST whose input and output alphabets are the same. In this case, both are the set of words.

Figure 2 shows an example of a WFSA containing different alternatives for the transcription of a utterance. This example shows a word-level graph that would represent a lattice. If it were a decoding graph, the arcs would be substituted by sub-graphs whose input labels would be senones and whose outputs labels would be words. For each sub-graph representing a word, the word is normally the output label of the first or the last arc in the sub-graph, whereas the output label for the rest of the arcs is an empty symbol, ϵ .

2.4 Language Model Re-scoring

Although it is possible to embed any language model in the decoding graph, it has a very important impact on the size of the decoding graph. Because of that, a common approach, described in

Fig. 2. Example of a word-level decoding graph.

References [10, 16, 30], is to embed a small language model in the decoding graph, extract a set of the best paths (lattice) from it, and recompute the weights (scores) of the lattice according to a larger language model. Finally, the best path is extracted from the re-scored lattice.

The language is normally modeled using n -grams, which are the probabilities for the next elements of the sequence given the previous $n - 1$ elements. N -grams express, in a simple and useful way, the relation between words in a sentence. But, also, they are easy to extract and are naturally represented using graphs and, thus, they can be easily converted to WFSTs. For this study, however, we followed a recent approach [15] that proposed the use of an RNN to replace the n -gram WFST. RNNs can potentially model unbounded word histories. However, replacing the n -gram WFST model with an RNN introduces some challenges. It is true that the histories are encoded in a very efficient way by the internal state of the network, but that state represents only one history, so for each of the histories explored during the re-scoring, the related internal states and the sequence it represents must be stored. The size of this table would grow really fast. To solve that, the authors propose to limit the length of the stored histories, merging together entries with $n - 1$ shared history elements, and thus greatly reducing memory usage in exchange for a small reduction in accuracy.

3 LOW-POWER ASR HARDWARE PLATFORM

In this section, we present a hardware platform that implements the human-quality ASR system described in Section 2. Our solution consists of a mobile CPU, a main DRAM memory, an accelerator to perform the Viterbi beam search, and another accelerator for DNN inference. Since specialized accelerators match the software closely and do not have the overheads of more generic circuits, they are very efficient, and thus, relying on them for the most compute intensive parts of the ASR system allows our solution to achieve real-time performance with a tiny power budget.

The main memory stores the data required (and generated) by the ASR, such as the TDNN and RNN weights, the decoding graph, and the different tables. The Viterbi accelerator carries out the decoding step entirely (Section 2.3), whereas the DNN accelerator performs the TDNN (Section 2.2) and RNN (Section 2.4) inference, as well as all the vector-matrix and matrix-matrix operation from the feature extraction step (Section 2.1), such as the Discrete Fourier Transform and the Cosine transform from the MFCC computation. By using these accelerators, the CPU is freed from the most compute intensive operations, relegating its role to the orchestration of the accelerators and the computation of the operations that are not well suited for them, e.g., some operations from the i -vector computation. Figure 3 shows a diagram of the ASR process, where each software component is linked to the hardware in which it is executed.

Fig. 3. High-level diagram of the complete system.

Fig. 4. Architecture of the DNN accelerator based on *DianNao* [3]. In addition to the Neural Function Unit (NFU), it includes three on-chip buffers to store inputs (*NBin*), weights (*SB*), and outputs (*NBout*). The main configuration parameter is T_n , which sets the number of parallel neurons and parallel synapses per neuron in the NFU. T_n also determines the port width of the memories.

3.1 DNN Accelerator

Since the main bottleneck of this system is the TDNN network, which is entirely composed of fully connected layers, others accelerators optimized for Convolutions or recurrent layers are not a good fit. Because of that, we decided to include an accelerator based on *DianNao* [3] (Figure 4). This accelerator is simple, and its power consumption and area are extremely low (25mW and 0.3mm, respectively), making it a very efficient option for our use case.

DianNao consists of a *Neural Functional Unit* (NFU) and some on-chip buffers. The NFU contains all the units required to perform the DNN computations, including an array of adders and multipliers, in addition to specialized units for the activation functions. The NFU is pipelined in three stages: NFU-1, to multiply the inputs by the weights; NFU-2, to add-reduce the results from

NFU-1; and NFU-3, to perform the activation function. Additionally, each NFU stage is pipelined to further increase the clock frequency.

As normally there are not enough multipliers in NFU-1 to compute all the inputs for the neuron (a neuron may have thousands of inputs, whereas, in hardware, we have in the order of tens of multipliers), NFU-3 is idle while NFU-1 and NFU-2 iterate through the inputs, accumulating the partial results. Only when all the inputs have been processed, NFU-3 computes the activation function for the neuron. To exploit more parallelism, the NFU contains resources to compute several neurons at the same time. Specifically, it computes Tn inputs for Tn neurons in parallel.

The internal memory in DianNao is composed of three SRAM memory buffers: *SB*, to store the weights; *NBin* for the inputs; and *NBout* to store the outputs. They each have an associated DMA to fetch input data in advance and thus hide the main memory latency and to send results back to main memory in background. Each cycle, the NFU requires Tn inputs and $Tn * Tn$ weights, and generates Tn outputs, which may or may not be stored on *NBout*. Because of that, the width for both *NBin* and *NBout* is Tn values, whereas the width of *SB* is $Tn * Tn$ values.

To allow the accelerator to be programmed, it contains a *Control Processor*, with an additional buffer to store instructions. The control processor fetches instructions from the instruction buffer, decodes them, and generates the control signals for the NFU and the memory requests for the DMAs of the different data buffers.

3.2 Viterbi Accelerator

The decoding step is a search over a directed, weighted graph. It begins from the *start node*, or *start state*, in the decoding graph. Then, the node is expanded by traversing its arcs, which are connected to several destination nodes. We compute the cost to reach each node by adding together the cost of traversing the arc, the acoustic score of the node (obtained by evaluating the acoustic model, i.e., the TDNN), and the accumulated cost from the source node. (Although the costs represent probabilities, additions are used instead of multiplications, since cost is computed as the negative log-likelihood.) For each frame, the acoustic model is evaluated and the set of active nodes, a.k.a. tokens, is expanded. To prevent the number of tokens to grow exponentially, a beam around the best current path is set: If the cost to reach a node is above that beam, it is discarded. This pruning of the search space largely reduces execution time with a negligible impact on accuracy.

The Viterbi search is a graph-processing algorithm that generates sparse and unpredictable memory accesses. On every iteration, it requires traversing a highly irregular WFST, expanding only a small and sparsely distributed subset of the nodes due to the pruning. Therefore, the Viterbi beam search is not well suited for execution on highly parallel hardware, such as GPUs or the previous DNN accelerator. Furthermore, it is not a good fit for the CPU either, as caches exhibit poor hit ratios due to the sparse memory accesses. To achieve high-performance and low-power Viterbi search, we included the accelerator described in Reference [37], which is specifically designed to execute the Viterbi beam search efficiently, in terms of energy and execution time.

Figure 5 illustrates the architecture of the Viterbi accelerator. It consists of several modules: *State Issuer* reads an active *token* and fetches the corresponding *state* in the decoding graph from main memory. The *Arc Issuer* receives the previously obtained state and fetches from main memory its output *arcs*. States and arcs are independently cached to exploit temporal locality. The *Acoustic Likelihood Issuer* receives the *senone IDs* related to the arcs obtained by the Arc Issuer and reads the corresponding *acoustic scores* from the *Acoustic Likelihood Buffer*, which is filled with the scores for the current frame, obtained from the Acoustic Model. The next component, the *Likelihood Evaluation*, computes the cost of traversing the arcs, which are sent to the *Token Issuer*, which discards them or creates the corresponding tokens to continue expanding the paths in the next frame, depending on whether the costs are within the *beam* or not. To store the tokens for the

Fig. 5. Architecture of the Viterbi accelerator. It consists of: several *Issuer* components to load/store data from/to the main memory, each with an associated cache memory; an additional module to compute the likelihood of the paths; and two *hash* memories to keep the active tokens for the current and the next frame.

current and the next frame, the accelerator contains two hash memories, which are swapped, with no memory transfers, at the beginning of the execution for each frame. If the tokens do not fit in the hash memory, they are sent to a reserved space in main memory labeled as *Overflow buffer*. Besides that, all the tokens for each frame are stored in another region of main memory, which can be used to obtain the single best sequence by backtracking from the best token at the last frame, or to obtain a word-level lattice. However, to obtain the lattice, instead of storing a single back-link to backtrack the best path, we have to store links to all the source tokens, representing a minor modification of the accelerator.

To deal with the sparse and irregular memory accesses, the Viterbi accelerator includes an area-effective solution based on the Decoupled Access-execute paradigm [31]. After the pruning step, the addresses of all the arcs that will be accessed in the current iteration, i.e., frame, can be computed in advance and memory requests can be issued early to tolerate the memory latency as described in Reference [37].

4 SYSTEM ANALYSIS

In this section, we describe the evaluation methodology and provide the experimental results that characterize our hardware-accelerated ASR solution.

Regarding our experimental setup, we configured the Viterbi accelerator with the parameters shown in Table 1. As compared to Reference [37], we shrank the size of the caches and hash tables, reducing the area from $24mm^2$ to $3.34mm^2$. This large reduction in on-chip memory has a small impact in performance, since our WFST for Viterbi search is significantly smaller than the one used in Reference [37]. Note that the accelerator in Reference [37] was designed for a single-pass ASR system that includes a more complex WFST for Viterbi search. However, our ASR system includes a re-scoring pass with an RNN-based language model after Viterbi search. Due to this re-scoring pass, the WFST used by Viterbi search can be largely reduced while maintaining the accuracy. Specifically, the WFST in Reference [37] contains 125K words and has a size of 618MB, compared to the 200K-word WFST used in this work, which requires only 181MB.

Table 1. Parameters for the Viterbi Accelerator

Technology	28 nm
Frequency	600 MHz
State Cache	128 KB, 4-way, 64 bytes/line
Arc Cache	256 KB, 4-way, 64 bytes/line
Token Cache	128 KB, 2-way, 64 bytes/line
Acoustic Likelihood Buffer	64 KB
Hash Table	768 KB, 32K entries
Memory Controller	32 in-flight requests
State Issuer	8 in-flight request
Arc Issuer	8 in-flight request
Token Controller	32 in-flight request
Acoustic Likelihood Issuer	1 in-flight arc
Likelihood Evaluation Unit	4 FP adders, 2 FP comparators

Table 2. Parameters for the DNN Accelerator

Technology	28 nm
Frequency	55 MHz
Bitwidth	1 Byte
Weight Buffer (SB)	64 entries, 16 KB
Input Buffer (NBin)	64 entries, 1 KB
Output Buffer (NBout)	64 entries, 1 KB
Tn	16 values

Table 3. Parameters for the CPU

Technology	28 nm
Frequency	1.7 GHz
CPU	ARM Cortex-A57
Number of Cores	4
Cache	32+48 KB L1, 2 MB shared L2

However, the DNN accelerator is configured as described in Table 2 ($Tn = 16$, 64 entries per buffer). To further reduce the size of the on-chip memories while keeping the throughput, the inputs and weights are quantized to 8 bits, which compresses its size by 4x (with respect to a baseline representation of 4-byte Floating Point) with a negligible impact on *WER*. Apart from that, we reduced the clock frequency from the 0.98GHz specified in the DianNao paper to 55MHz. Because of using such a low clock rate, our NFU stages are not pipelined. This frequency allows for real-time execution while largely reducing the power and memory bandwidth requirements, making the solution more amenable for low-power mobile systems. The area of the DNN accelerator is $0.3mm^2$ in 28 nm technology. Finally, the mobile CPU included in our system is a low-power Quad-Core ARM whose parameters are shown in Table 3, whereas the main memory consists of 8 GB of LPDDR4.

Table 4. WER Obtained by Different State-of-the-Art Systems on the Librispeech Corpus

system	E2E	LM	test clean	test other
Human	-	-	5.83	12.69
DeepSpeech2 [1]	Yes	5-gram	5.15	12.73
wav2letter++ [39]	Yes	Conv	3.44	11.24
Jasper DR 10x5 [13]	Yes	Transformer-XL	2.95	8.79
pFSMN-Chain [36]	No	TDNN-LSTM	2.97	7.5
Kaldi+Quant	No	TDNN-LSTM	3.67	9.76

To evaluate the performance of the different components of the system, we retrieved information from several sources. Measurements of the CPU execution time and energy were obtained from hardware performance counters. However, we used a cycle-level simulator that accurately tracks the architecture of the Viterbi and DNN accelerators to obtain their respective execution times.

We obtained the power dissipation and delay of the critical path for the DNN and Viterbi accelerators by using different tools. First, we used CACTI to estimate area, energy consumption, and access time for the on-chip memories of the accelerators. Second, we implemented the different pipeline components in Verilog and synthesized them using Synopsys Design Compiler. The maximum frequency was set according to the minimum time required to propagate the signal through the logic components and memories, as reported by Synopsys Design Compiler and CACTI, respectively. To estimate total energy consumption, we used the activity factors from the cycle-level simulators and the energy cost of each operation and memory access from Synopsys Design Compiler and CACTI. The main memory was modeled using the *Micron TN5301_LPDDR4 System Power Calculator* [18] with the parameters from the Micron's Z91M package.

The next subsections provide an analysis of the recognition accuracy, memory requirements, performance, and power dissipation of the hardware-accelerated ASR system.

4.1 Recognition Accuracy

Accuracy in ASR is measured in Word Error Rate (WER), which represents the distance between the decoded sentence and the correct one. It is computed as the sum of the number of additions, deletions, and substitutions required to transform the decoded sentence into the reference one, divided by the number of words in the reference sentence. To enhance reproducibility, ASR systems are trained and tested against standard speech corpora that are commonly used by the ASR community. Each speech corpus is categorized by some of the properties of the utterances contained in it, such as whether they consist of recorded conversations or read text, the level of noise, the language, or the variety of speakers. The current approach is to train the systems for a specific context, as opposed to training a general system. Hence, to test a system against each corpus, it has to be retrained using utterances from the specific corpus.

The system used for this study was trained for the *Librispeech* speech corpus. Librispeech is divided into five sets: *train*, *dev_clean*, *test_clean*, *dev_other*, and *test_other*. All but *train* are normally used to test the system. *dev_other* and *test_other* contain difficult-to-decode utterances, either because they are noisy or because the accent is strong. Table 4 shows the WER of several systems proposed in the literature. The first row corresponds to the human performance, according to Reference [1]. The following three rows are E2E systems, whereas pFSMN-Chain and Kaldi are hybrid DNN-Viterbi solutions.

Although pFSMN-Chain outperforms Kaldi, we decided to use Kaldi instead, because they are similar in almost every aspect, both of them being implemented on the Kaldi framework. The main

Table 5. Memory Requirements for the Models of the ASR System

Component	Memory (MB)	Percentage (%)
MFCC	0.68	0.07
I-vector	17.95	1.77
Decoding	180.8	17.85
Acoustic Model	16.17	1.6
Language Model	16.12	1.59
Word Embeddings	781.28	77.13
TOTAL	1,013	100

difference between them is that pFSMN-Chain employs a special CNN network as Acoustic Model, while Kaldi relies on a TDNN network. As the results obtained with Kaldi are very competitive and the network is simpler, we believe it is a good representative of a state-of-the-art hybrid system.

4.2 Memory

The hybrid ASR system we are presenting employs two neural networks: a TDNN for acoustic model and a TDNN-LSTM for language model. Furthermore, it requires a decoding graph (WFST) and the models related to i-vector and MFCC computation.

Table 5 shows the amount of memory required by each component and the percentage of the total that it represents. To reduce the memory footprint, all the weights from the TDNN and the LSTM network are quantized to 8 bits with negligible accuracy loss in WER. The inputs to all the layers are also quantized.

The component with the largest impact on memory footprint is the *Word Embedding Table*, which is used to generate an embedding representation of the words that is more effective for the language model re-scoring phase, which is implemented through an RNN.

Even more important than memory footprint is the required traffic between main memory and the accelerators. Although the sizes of the neural networks are relatively small, they are not small enough so they can be stored in on-chip memory, so they must be kept in main memory and read entirely for each inference. The bit rate required between main memory and the DNN accelerator is going to be one of the limiting factors of the system, imposing a maximum useful clock frequency. In our model, the neural network accelerator is configured at 55MHz, requiring 13.3 GB/s from the memory, which represents the 80% of its maximum bandwidth of 16,800 MB/s. The bit rate required by the Viterbi accelerator is much lower, ranging from 1.9 to 7.5 GB/s (11.5% and 45% of the maximum memory bandwidth, respectively), depending on the utterance.

4.3 Execution Time

We have measured execution time and *Real Time Factor* (RTF) by computing the time required to execute each component of the ASR system on the hardware it is mapped to. All the parts mapped to the CPU were measured directly by internal counters. Parts of the ASR system mapped to the accelerators were simulated to obtain cycle count, and then the number of cycles was multiplied by the cycle time of the specific accelerator to obtain execution time.

To analyze the gains achieved by using a custom hardware, three alternative architectures were studied and compared. In the following plots, *CPU* refers to executing all the process on the CPU, *CPU-GPU* executes all the highly parallel computations on the GPU (TDNN and RNN), and *CPU-ACCEL* includes the aforementioned accelerators to execute the Viterbi search, the TDNN, and RNN computations.

Fig. 6. Cumulative distribution of the RTF for all utterances in the test set. The plots correspond to the execution of the Kaldi system on the three hardware architectures: CPU, CPU-GPU, and CPU-ACCEL.

Fig. 7. Execution time breakdown for the different ASR components obtained on the three hardware architectures. For each architecture, the three bars represent the utterances at percentiles 0, 50, and 100 in the RTF plot (Figure 6).

Figure 6 shows the RTF distribution among the utterances, plotted as cumulative frequency. The x axis is the RTF, whereas the y axis is the percentage of utterances decoded on an RTF lower than x . In RTF, lower is better, and an RTF lower than one means that the utterance is processed faster than in real-time. We can draw two main conclusions from Figure 6. First, using custom hardware provides an important performance improvement (around $4.5x$ improvement compared to the GPU-based system), and it is key to efficiently guarantee real-time for all utterances. Second, most of the utterances lay in a narrow region of RTF, especially for the *CPU-ACCEL* system, with some important outliers (around 10% of the test utterances in our experiments) laying very far from that region. A direct consequence of this high variability is that a system dimensioned to guarantee a specific performance for the worst case would be highly oversized for most of the utterances. However, these outliers represent real scenarios that cannot be ignored.

To study the bottlenecks in execution time, several utterances were chosen to represent the different regions in the previous figure. More specifically, we sorted the utterances in ascending order of RTF and chose those located at the percentiles 0, 50, and 100. Figure 7 shows the percentage of execution time dedicated to each ASR component. The x axis includes the three

Fig. 8. Power dissipation during the computation of the ASR components. Each bar represents the power dissipated by each ASR component, broken down by hardware subsystem.

systems mentioned above: *CPU*, *CPU-GPU*, and *CPU-ACCEL*. For each system, there are three bars representing the breakdown for the selected utterances: percentiles 0, 50, and 100. We can observe that the main bottleneck for most of the cases is the TDNN inference, computed on the DNN accelerator. TDNN computation is limited by the main memory bandwidth as mentioned in Section 4.2. For the case of utterances represented by the 100% percentile, the language model RNN (RNNLM) inference is the main bottleneck, also computed on the DNN accelerator. This can be explained by looking at the size of the lattice generated by Viterbi for the previous utterances: 326B, 29KB and 281KB. The lattice for the utterance at the percentile 100% required 1,507 RNN evaluations, whereas the lattice for the utterance at 0% required only 1 RNN evaluation. Utterances like the one at the percentile 100% are most difficult to decode. Because of that, the Viterbi search has to explore a larger number of alternative paths, generating a larger lattice. The difficulty of decoding an utterance has an important impact on LM re-scoring, which makes this component the main source of the RTF variability shown in Figure 6.

4.4 Power Consumption

Figure 8 shows the average power dissipated by the ASR system broken down into the main ASR components. As we can see, the peak power is very close to 2.5W, reached during the computation of the i-vector. Note that computing the i-vector requires the use of the CPU, the most power-demanding hardware component in our system, whereas for the rest of the time the CPU is mostly idle. During the Viterbi computation, the DNN accelerator is power-gated, so the bar shows the power dissipated by the Viterbi accelerator, the CPU in idle mode, and the main memory, resulting in 519mW. The rest of the components, i.e., MFCC, TDNN, and RNNLM, are computed almost exclusively on the DNN accelerator, while the Viterbi accelerator is power-gated and the CPU is idle, so the corresponding bars show the power dissipated by the DNN accelerator, the CPU in idle mode, and the memory. The average power is slightly above 1W, where most of it (about 95%) is due to the main memory, which is intensively accessed (4.2).

Regarding the energy consumption, our results show that decoding in our platform requires 4.3x less energy per frame than using the CPU-GPU system (Figure 9). As shown in Figure 10, most of the energy (71.3%) consumed by our platform is due to main memory: 26.5% is consumed by the CPU, and the rest (less than 3%) is consumed by the accelerators.

All the above results show that the proposed ASR system can be integrated in low-power devices due to its low area and power budget. Besides, note that this evaluation has been performed using a

Fig. 9. Cumulative distribution of the energy per frame for decoding for all utterances in the test set. The plot shows the energy for the execution on the three hardware platforms: CPU, CPU-GPU, and CPU-ACCEL.

Fig. 10. Breakdown of the energy consumed during the decoding of an average utterance on our accelerated platform.

28nm technology process due to the available tools. In a more up-to-date technology process (e.g., 10 or 7nm) both power and area would be significantly lower (by around one order of magnitude).

5 RELATED WORK

Prior research on hardware-accelerated speech recognition was focused on older ASR systems and assumed smaller vocabularies and/or acoustic models than state-of-the-art solutions as the one considered in this article. Price et al. [27] designed a chip encompassing from *Voice Activity Detection* (VAD) and audio capture, to Viterbi-based WFST decoding. The area of the chip is 13.18mm^2 and consumes 11.27mW (not including power from off-chip components, such as main memory, which is the main bottleneck according to our models) while running a 145K word vocabulary benchmark. However, our work focused on much bigger models, including a 200K vocabulary decoding graph, a bigger acoustic model (16.17MB, as opposed to their 3.71MB model) and a language model re-scoring step based on LSTM to achieve state-of-the-art performance.

Yazdani et al. [37, 38] propose a system based on a Viterbi accelerator and a GPU for DNN inference, consuming 462mW plus the GPU power (between 2W and 6W). Our work builds upon that system by adding a DianNao-based accelerator to replace the GPU and re-configuring the Viterbi accelerator to meet our target RTF in the smallest possible area. Our work also differs in that we use a more sophisticated ASR system based on larger and more accurate speech models. More specifically, the ASR system used in References [37, 38] achieves 10.62% WER for Librispeech test clean dataset, whereas our system delivers 3.67% WER for the same audio files.

Earlier proposals for hardware-accelerated ASR [4, 14, 19] focused on GMM-based recognizers, with CMU's *Sphinx* as a usual software baseline, and vocabularies with less than 100K words (e.g.,

5K/20K-word *Wall Street Journal*, 64K-word Broadcast News). More recently, Tabani et al. [32] proposed an accelerator for the *PocketSphinx* system, configured to decode a 130K-word librispeech-based benchmark. *PocketSphinx* is based on *CMU Sphinx*, aimed at portability. By using that accelerator (a 0.94mm^2 , 110mW chip), the decoding time and energy is reduced by $5.89\times$ and $241\times$, respectively, over a mobile GPU implementation. However, this type of systems have become less popular nowadays due to their lower accuracy. For instance, Tabani et al. [32] report a WER of 24.14, which is much higher than current state-of-the-art systems as those reported in Table 4.

6 CONCLUSIONS

Automatic Speech Recognition is becoming a key technology for a large variety of computing devices. With users getting used to this kind of interface, and the technology improving every day, we can easily foresee a world in which this technology occupies a central role in human-machine interactions. In this article, we have presented and evaluated a human-quality ASR system and proposed a hardware platform that provides real-time recognition while consuming around 1W of average power. The key to achieve real-time and low-power results is the use of two hardware accelerators (accounting for a total area of 3.64mm^2) for the most compute-intensive components of the ASR system: Viterbi beam search and DNN evaluation. Overall improvement over a low-power GPU-based system is a $4.5\times$ speedup and a $4.3\times$ reduction in energy consumption per frame. These results show that it is possible to implement a high-performance ASR system that runs locally on low-power devices without the need for server-based ASR services on the cloud.

REFERENCES

- [1] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. 2016. Deep Speech 2: End-to-end speech recognition in English and Mandarin. In *Proceedings of the International Conference on Machine Learning*. 173–182.
- [2] Herve A. Bourlard and Nelson Morgan. 2012. *Connectionist Speech Recognition: A Hybrid Approach*. Vol. 247. Springer Science & Business Media.
- [3] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM Sigplan Notices*, Vol. 49. ACM, 269–284.
- [4] Anthony Chun, Jenny X. Chang, Zhen Fang, Ravishankar Iyer, and Michael Deisher. 2011. ISIS: An accelerator for Sphinx speech recognition. In *Proceedings of the IEEE 9th Symposium on Application Specific Processors (SASP'11)*. IEEE, 58–61.
- [5] Najim Dehak, Patrick J. Kenny, Réda Dehak, Pierre Dumouchel, and Pierre Ouellet. 2011. Front-end factor analysis for speaker verification. *IEEE Trans. Audio, Speech Lang. Proc.* 19, 4 (2011), 788–798.
- [6] V. V. Digalakis, D. Rtischev, and L. G. Neumeyer. 1995. Speaker adaptation using constrained estimation of Gaussian mixtures. *IEEE Trans. Speech Audio Proc.* 3, 5 (1995), 357–366.
- [7] Alex Graves and Navdeep Jaitly. 2014. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*. 1764–1772.
- [8] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. 2013. Hybrid speech recognition with deep bidirectional LSTM. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*. IEEE, 273–278.
- [9] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. 2012. Deep neural networks for acoustic modeling in speech recognition. *IEEE Sig. Proc. Mag.* 29 (2012).
- [10] Takaaki Hori, Chiori Hori, Yasuhiro Minami, and Atsushi Nakamura. 2007. Efficient WFST-based one-pass decoding with on-the-fly hypothesis rescoring in extremely large vocabulary continuous speech recognition. *IEEE Trans. Audio, Speech Lang. Proc.* 15, 4 (2007), 1352–1365.
- [11] Takaaki Hori and Atsushi Nakamura. 2005. Generalized fast on-the-fly composition algorithm for WFST-based speech recognition. In *Proceedings of the 9th European Conference on Speech Communication and Technology*.
- [12] Martin Karafiát, Lukáš Burget, Pavel Matějka, Ondřej Glembek, and Jan Černocký. 2011. iVector-based discriminative adaptation for automatic speech recognition. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition & Understanding*. IEEE, 152–157.

- [13] Jason Li, Vitaly Lavrukhin, Boris Ginsburg, Ryan Leary, Oleksii Kuchaiev, Jonathan M. Cohen, Huyen Nguyen, and Ravi Teja Gadde. 2019. Jasper: An end-to-end convolutional neural acoustic model. *arXiv preprint arXiv:1904.03288* (2019).
- [14] Edward C. Lin, Kai Yu, Rob A. Rutenbar, and Tsuhan Chen. 2006. Moving speech recognition from software to silicon: The in silico vox project. In *Proceedings of the 9th International Conference on Spoken Language Processing*.
- [15] Xunying Liu, Yongqiang Wang, Xie Chen, Mark J. F. Gales, and Philip C. Woodland. 2014. Efficient lattice rescoring using recurrent neural network language models. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'14)*. IEEE, 4908–4912.
- [16] Andrej Ljolje, Fernando Pereira, and Michael Riley. 1999. Efficient general lattice generation and rescoring. In *Proceedings of the 6th European Conference on Speech Communication and Technology*.
- [17] Yajie Miao, Mohammad Gowayyed, and Florian Metze. 2015. EESSEN: End-to-end speech recognition using deep RNN models and WFST-based decoding. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU'15)*. IEEE, 167–174.
- [18] Micron Technology Inc. 2016. TN-53-01: LPDDR4 System Power Calculator. urlRetrieved from <https://www.micron.com/support/tools-and-utilities/power-calc>.
- [19] Kazuo Miura, Hiroki Noguchi, Hiroshi Kawaguchi, and Masahiko Yoshimoto. 2008. A low memory bandwidth Gaussian Mixture Model (GMM) processor for 20,000-word real-time speech recognition FPGA system. In *Proceedings of the International Conference on Field-programmable Technology*. IEEE, 341–344.
- [20] Mehryar Mohri, Fernando Pereira, and Michael Riley. 2002. Weighted finite-state transducers in speech recognition. *Comput. Speech Lang.* 16, 1 (2002), 69–88.
- [21] Mehryar Mohri, Fernando Pereira, and Michael Riley. 2008. Speech recognition with weighted finite-state transducers. In *Springer Handbook of Speech Processing*. Springer, 559–584.
- [22] Stefan Ortmanns, Hermann Ney, and Xavier Aubert. 1997. A word graph algorithm for large vocabulary continuous speech recognition. *Comput. Speech Lang.* 11, 1 (1997), 43–72.
- [23] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. 2015. Librispeech: An ASR corpus based on public domain audio books. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'15)*. 5206–5210. DOI: <https://doi.org/10.1109/ICASSP.2015.7178964>.
- [24] Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. 2015. A time delay neural network architecture for efficient modeling of long temporal contexts. In *Proceedings of the 16th Conference of the International Speech Communication Association*.
- [25] Gian Paolo Perrucci, Frank H. P. Fitzek, and Jörg Widmer. 2011. Survey on energy consumption entities on the smartphone platform. In *Proceedings of the IEEE 73rd Vehicular Technology Conference (VTC Spring'11)*. IEEE, 1–6.
- [26] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. 2011. The Kaldi speech recognition toolkit. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society.
- [27] Michael Price et al. 2016. *Energy-scalable Speech Recognition Circuits*. Ph.D. Dissertation. Massachusetts Institute of Technology.
- [28] Lawrence R. Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* 77, 2 (1989), 257–286.
- [29] L. R. Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* 77, 2 (Feb. 1989), 257–286. DOI: <https://doi.org/10.1109/5.18626>
- [30] Haşim Sak, Murat Saraclar, and Tunga Güngör. 2010. On-the-fly lattice rescoring for real-time automatic speech recognition. In *Proceedings of the 11th Conference of the International Speech Communication Association*.
- [31] James E. Smith. 1982. Decoupled access/execute computer architectures. In *ACM SIGARCH Computer Architecture News*, Vol. 10. IEEE Computer Society Press, 112–119.
- [32] Hamid Tabani, Jose-Maria Arnau, Jordi Tubella, and Antonio Gonzalez. 2017. An ultra low-power hardware accelerator for acoustic scoring in speech recognition. In *Proceedings of the 26th International Conference on Parallel Architectures and Compilation Techniques (PACT'17)*. IEEE, 41–52.
- [33] Alexander Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J. Lang. 1989. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37, 3 (1989), 328–339.
- [34] Hisashi Wakita. 1977. Normalization of vowels by vocal-tract length and its application to vowel identification. *IEEE Trans. Acoust. Speech Sig. Proc.* 25, 2 (1977), 183–192.
- [35] Hainan Xu, Tongfei Chen, Dongji Gao, Yiming Wang, Ke Li, Nagendra Goel, Yishay Carmiel, Daniel Povey, and Sanjeev Khudanpur. 2018. A pruned RNNLM lattice-rescoring algorithm for automatic speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'18)*. IEEE, 5929–5933.

- [36] Xuerui Yang, Jiwei Li, and Xi Zhou. 2018. A novel pyramidal-FSMN architecture with lattice-free MMI for speech recognition. *arXiv preprint arXiv:1810.11352* (2018).
- [37] Reza Yazdani, Albert Segura, Jose-Maria Arnau, and Antonio Gonzalez. 2016. An ultra low-power hardware accelerator for automatic speech recognition. In *Proceedings of the 49th IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*. IEEE, 1–12.
- [38] Reza Yazdani, Albert Segura, Jose-Maria Arnau, and Antonio Gonzalez. 2017. Low-power automatic speech recognition through a mobile GPU and a Viterbi accelerator. *IEEE Micro* 37, 1 (2017), 22–29.
- [39] Neil Zeghidour, Qiantong Xu, Vitaliy Liptchinsky, Nicolas Usunier, Gabriel Synnaeve, and Ronan Collobert. 2018. Fully convolutional speech recognition. *arXiv e-prints*, Article arXiv:1812.06864 (Dec 2018).
- [40] Fang Zheng, Guoliang Zhang, and Zhanjiang Song. 2001. Comparison of different implementations of MFCC. *J. Comput. Sci. Technol.* 16, 6 (2001), 582–589.

Received May 2020; revised September 2020; accepted September 2020