The Dark Side of DNN Pruning

Reza Yazdani, Marc Riera, Jose-Maria Arnau, Antonio González
Department of Computer Architecture
Universitat Politécnica de Catalunya (UPC), Barcelona, Spain
{ryazdani, mriera, jarnau, antonio}@ac.upc.edu

***Abstract*—DNN pruning has been recently proposed as an effective technique to improve the energy-efficiency of DNN-based solutions. It is claimed that by removing unimportant or redundant connections, the pruned DNN delivers higher performance and energy-efficiency with negligible impact on accuracy. However, DNN pruning has an important side effect: it may reduce the confidence of DNN predictions. We show that, although top-1 accuracy may be maintained with DNN pruning, the likelihood of the class in the top-1 is significantly reduced when using the pruned models. For applications such as Automatic Speech Recognition (ASR), where the DNN scores are consumed by a successive stage, the workload of this stage can be dramatically increased due to the loss of confidence in the DNN.**

An ASR system consists of a DNN for computing acoustic scores, followed by a Viterbi beam search to find the most likely sequence of words. We show that, when pruning the DNN model used for acoustic scoring, the Word Error Rate (WER) is maintained but the execution time of the ASR system is increased by 33%. Although pruning improves the efficiency of the DNN, it results in a huge increase of activity in the Viterbi search since the output scores of the pruned model are less reliable.

Based on this observation, we propose a novel hardware-based ASR system that effectively integrates a DNN accelerator for pruned models with a Viterbi accelerator. In order to avoid the aforementioned increase in Viterbi search workload, our system loosely selects the N-best hypotheses at every time step, exploring only the N most likely paths. To avoid an expensive sort of the hypotheses based on their likelihoods, our accelerator employs a set-associative hash table to keep track of the best paths mapped to each set. In practice, this solution approaches the selection of N-best, but it requires much simpler hardware. Our approach manages to efficiently combine both DNN pruning and Viterbi search, and achieves 9x energy savings and 4.2x speedup with respect to the state-of-the-art ASR solutions.

***Keywords*-Deep Learning; DNN Pruning; Automatic Speech Recognition (ASR); Viterbi Search; Hardware Accelerator;**

I. Introduction

DNN pruning has attracted the attention of the architectural community in recent years [1], [2], [3], [4], [5], [6]. Based on the observation that DNN models tend to be oversized and include a high degree of redundancy, pruning aims at reducing the model size by identifying and removing unimportant connections. The pruned model is claimed to retain accuracy, while it requires much smaller memory storage and significantly less computations, resulting in large performance improvements and energy savings.

The caveat here is how to measure accuracy. These previous works on DNN pruning employ the top-1 or top-5 as the accuracy metric. For top-1 accuracy, the output of the pruned model is considered to be correct if the output with maximum likelihood, i.e. the top-1 class, corresponds to the correct class. However, in the context of an Automatic Speech Recognition (ASR) system, it is crucial to take into account the likelihood, a.k.a. score, assigned to the top-1 class. We refer to this likelihood as the confidence of the DNN prediction. In this paper, we show that DNN pruning has a large impact on the confidence of DNN predictions, resulting in much lower likelihood for the top-1 class.

In speech recognition, the DNN is used not only to label objects with a particular class, but the probability that this class is correct (i.e., the score computed by the DNN) is used for further calculations. Figure 1 illustrates this problem with a state-of-the-art DNN for speech recognition [7]. This DNN generates the likelihoods for 3482 classes, that correspond to different sub-phonemes of the language. Figure 1 shows the distribution of the DNN scores for the original DNN and three pruned models at 70%, 80% and 90% of pruning, using a state-of-the-art pruning scheme proposed by Han et al. [1]. As it can be seen, the top-1 class (i.e. the peak likelihood) is the same for all the models. However, the distribution of likelihoods is severely affected by the pruning. The original model's prediction has a very high confidence of 0.92, i.e. the DNN excels at discriminating the correct class from the incorrect ones. On the contrary, the confidence of the pruned models is largely reduced, being lower than 0.5 for all of them and as low as 0.17 for 90% of pruning, resulting in a much less reliable prediction. Despite being admittedly a well selected example, this behavior is actually quite prevalent. We have observed that, for five hours of speech (1.8 million DNN executions), the confidence of the pruned DNN at 90% of pruning is reduced by more than 20% on average.

In this paper, we show that this reduction in DNN confidence has a high impact on the performance and energy consumption of an ASR system. In ASR, the input audio is split in frames of, typically, 10 ms of speech. Next, these frames are fed to a DNN that computes the acoustic scores, i.e. the probabilities of the different sub-phonemes in the language, for every frame of speech. Finally, the DNN scores are used to perform a Viterbi beam search to find the most likely sequence of words. Figure 2 shows the execution time and accuracy of a state-of-the-art ASR system that employs a DNN accelerator and a Viterbi search accelerator. When applying pruning, accuracy is largely maintained, and it is only affected for very aggressive pruning (90%). Regarding the execution time, although pruning improves the performance of the DNN

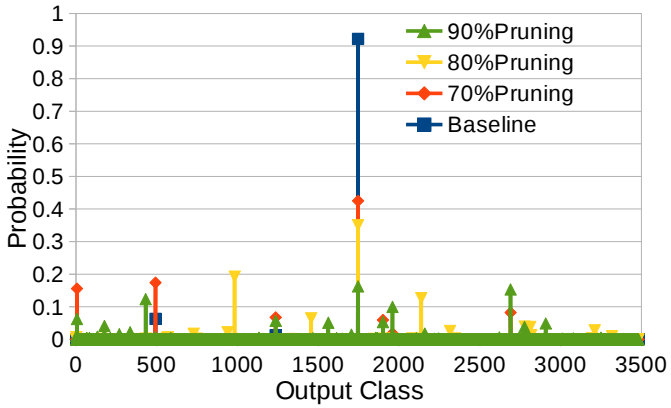

Fig. 1. Distribution of scores for a DNN for speech recognition and three pruned version at 70%, 80% and 90% of pruning. Although pruned models correctly identify top-1 class, the distribution of likelihoods is severely affected and confidence is largely reduced.

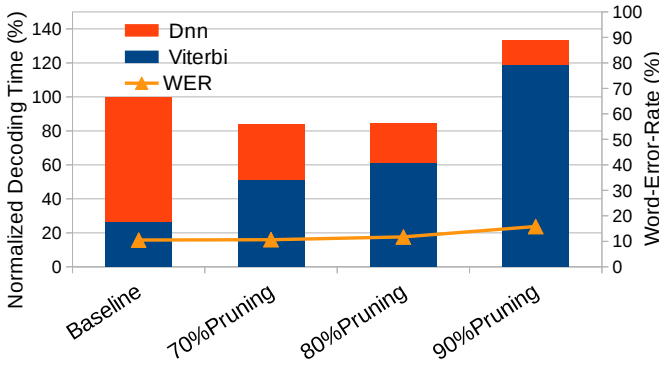

Fig. 2. Normalized execution time and Word Error Rate (WER) for a state-of-the-art ASR system, using a DNN with different degrees of pruning: 0% (Baseline), 70%, 80% and 90%. Pruning has a large impact on the execution time of the Viterbi beam search.

accelerator, it introduces a large slowdown in the Viterbi beam search, i.e. the consumer of the DNN scores. The increase in Viterbi search execution time offsets part of the benefits of the DNN pruning, and it results in 33% slowdown for 90% of pruning.

The Viterbi search suffers a slowdown of up to 4.5x when using DNN pruning. The reason for this slowdown is the aforementioned loss in DNN confidence. Viterbi beam search explores multiple interpretations of the input speech, but it discards very unlikely hypotheses, a.k.a. paths, to keep the search space manageable. Hypotheses whose distance with respect to the best path is larger than a given threshold are discarded (this is known as beam search and is used by all systems since it is unfeasible to explore all alternative paths). When using the original DNN, the paths using correct sub-phonemes (classes in top-1) get very high scores whereas the hypotheses using incorrect sub-phonemes obtain low scores and are discarded by the beam, reducing the search space. However, with the pruned DNN, the difference between the best path and the rest is significantly reduced and, hence, many more hypotheses fall within the beam distance and are explored. In other words, if the DNN clearly identifies the sub-phoneme for a frame of speech (like Baseline in Figure 1), only words that include that sub-phoneme are considered. On the contrary, if the DNN is less confident and assigns similar likelihood to multiple sub-phonemes (like 90% Pruning in Figure 1), more paths are explored, resulting in the slowdowns illustrated in Figure 2.

After analyzing and quantifying the side effects of pruning, the second part of this work investigates how this negative effect can be avoided for speech recognition. The straightforward solution is to reduce the beam width to discard more paths, mitigating the effects of the unreliable DNN. However, we observed that this solution suffers from long tail latencies, which is undesirable for real-time systems, as some utterances still suffer the workload increase. Instead, we take a completely different approach and propose to extend the hardware of the Viterbi accelerator to loosely track the N-best paths at every time step. By fixing the value of N, we restrict the search space avoiding the workload explosion, but we retain accuracy as we explore the most promising paths. Our proposal only requires a small hash table that keeps track of the best paths mapped to each set, avoiding an expensive global sort of all the hypotheses. We show that this solution is comparable to a system that accurately identifies the N-best paths, but it requires much simpler hardware and is faster. Our experimental results show that this scheme efficiently integrates DNN pruning with Viterbi beam search, outperforming the state-of-the-art solutions for ASR acceleration.

In this paper, we focus on the interaction between DNN pruning and Viterbi beam search, and its importance in the design of high-performance and energy-efficient ASR systems. We claim the following contributions:

- We make the observation that DNN pruning has a negative aspect that can be very important for speech recognition: it significantly reduces the confidence of DNN predictions by more than 20%.
- We show that the reduction in DNN confidence has a large impact on performance and energy-efficiency of an ASR system. The pruned DNN increases the execution time of the Viterbi beam search by up to 4.5x and, in fact, aggressively pruned DNNs result in worse global performance.
- We propose a novel scheme that effectively combines DNN pruning and Viterbi beam search for ASR. Our proposal requires simple changes to the hardware of a Viterbi accelerator. Our results show that our system outperforms the state-of-the-art in hardware-accelerated ASR, achieving 4.2x speedup and 9x energy savings.

II. Negative Effects of DNN pruning

In this section, we analyze the impact of DNN pruning on the quality of the predictions of a state-of-the-art DNN for speech recognition. Unlike prior work, we employ more complete metrics to assess the quality of the pruned models, considering not only the top-1 or top-5 error, but also the

TABLE I

KALDI'S DNN FOR SPEECH RECOGNITION. FC, P AND N STAND FOR FULLY-CONNECTED, POOLING AND NORMALIZATION RESPECTIVELY. THE TABLE ALSO INCLUDES THE NUMBER OF NEURONS AND WEIGHTS IN EACH LAYER, AND THE PERCENTAGE OF PRUNING AT EACH LAYER FOR GLOBAL PERCENTAGES OF 70%, 80% AND 90%. NOTE THAT FC0 IS NOT TRAINABLE IN KALDI AND, HENCE, IT CANNOT BE PRUNED.

Layer	FC0	FC1	P1	N1	FC2	P2	N2	FC3	P3	N3	FC4	P4	N4	FC5	SoftMax
Neurons	360	2000	400	400	2000	400	400	2000	400	400	2000	400	400	3482	3482
Weights	129k	720k	0	0	800k	0	0	800k	0	0	800k	0	0	1.4M	0
Pruning (70%)	0	71%	-	-	68%	-	-	65%	-	-	95%	-	-	66%	-
Pruning (80%)	0	82%	-	-	80%	-	-	77%	-	-	98%	-	-	78%	-
Pruning (90%)	0	92%	-	-	92%	-	-	91%	-	-	99%	-	-	90%	-

confidence of the DNN in its predictions. Finally, we describe the importance of DNN confidence in the context of ASR systems, characterizing its impact in the workload of the Viterbi beam search.

A. DNN Pruning

Modern DNNs contain a large number of parameters that require considerable storage and memory bandwidth. This hinders the deployment of DNNs in energy-constrained devices such as Smartphones or Tablets. However, DNNs tend to be over-dimensioned, and they typically exhibit significant redundancy [8]. Machine learning practitioners tend to oversize their models to guarantee superior prediction accuracy. Therefore, with a proper strategy, it is possible to compress the DNNs without significantly losing accuracy. DNN pruning [6], [4], [1] appears to be one of the most successful techniques for reducing model size. Pruning largely reduces DNN size, requiring significantly less storage and computational resources.

In this paper, we implement a state-of-the-art pruning scheme proposed by Han et al. [1] and analyze its impact in the accuracy and confidence in the context of a state-of-the-art ASR system. This pruning technique consists of three steps. First, the DNN is trained from scratch by using the conventional network training. Once the DNN is trained, low-weight connections are pruned, i.e. all connections whose weight is below a threshold are removed from the model. The threshold is the result of multiplying the standard deviation of the layer's weights by a quality parameter, as described in [1]. This quality parameter is used to control the degree of pruning. Finally, the pruned DNN is retrained to learn the final weights for the remaining connections.

B. Confidence of the Pruned DNN

We analyze in this section the impact of pruning in a state-of-the-art DNN for speech recognition. We prune the DNN proposed in [7] that is implemented in Kaldi [9], a popular ASR toolkit widely employed in academia and industry. Kaldi's DNN is a Multi Layer Perceptron (MLP), which consists of multiple fully-connected layers interleaved with pooling and normalization layers, as shown in Table I. This DNN takes as input the acoustic features for nine frames of speech: the current frame, the previous four frames and the next four frames. Each frame of speech is represented as a vector of 40 features. Therefore, the input of the DNN is a vector of 360 acoustic features.

The first layer of the DNN is a non-trainable fully-connected layer. Its weights are fixed before training in order to implement Linear Discriminant Analysis (LDA) [10]. Next, the DNN includes four fully-connected hidden layers, that include pooling and normalization. Finally, the output layer is fully-connected with a softmax activation function that generates the final likelihoods. The DNN generates the acoustic scores for 3482 sub-phonemes in the language. Table I also shows the number of neurons and weights in each layer. Kaldi's DNN contains more than 4.5 million learnable parameters.

We apply to this DNN the pruning technique presented by Han et al. as described in [1]. We set the threshold for each layer by multiplying a quality parameter, that is shared by all the layers, by the standard deviation of the weights in the given layer. To achieve a global pruning of 70%, 80% and 90%, we use a quality parameter of 1.44, 1.90 and 2.71 respectively. Table I reports the degree of pruning applied to each layer. Note that the first fully-connected layer, FC0, cannot be pruned as its weights are fixed to implement LDA. We disable the pruning in this layer as we cannot retrain it to recover accuracy, and we have to preserve LDA functionality [10]. Nevertheless, its weights are accounted for the total model size. The original non-pruned DNN is trained using the training dataset from LibriSpeech [11]. This training dataset is also used for the retraining after the pruning.

We have evaluated the top-5 accuracy for these pruned models, for more than five hours of speech from LibriSpeech test set, and found that the difference in top-5 error with respect to the non-pruned model is smaller than 3% for 70% and 80% pruning, and smaller than 5% for 90% pruning. In addition to the top-5 error, we have also evaluated the impact on DNN confidence. We measure the confidence as the probability assigned to the class with maximum likelihood, i.e. the top-1 class. Figure 3 shows the impact of pruning on DNN confidence. For the non-pruned DNN, the average confidence of the prediction is 0.68, i.e. the likelihood assigned to the top-1 class by the DNN is 0.68 on average. However, for 70% of pruning confidence is reduced to 0.65 (5% drop), whereas the model pruned at 80% exhibits a confidence loss of 9%. Finally, for 90% of pruning the average confidence in DNN predictions is reduced to 0.53, that represents a 22% drop in confidence.

Although the top-5 accuracy drops by less than 5% for 90% pruning, DNN confidence is reduced by 22%. In the next subsection, we show that this loss in confidence has a high impact on the performance of an ASR system, as it produces

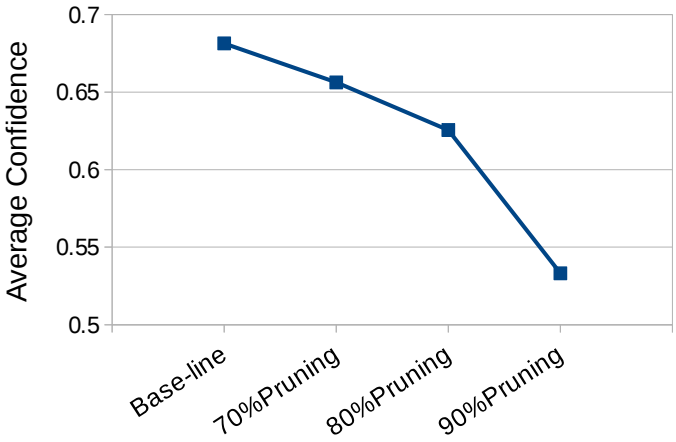

Fig. 3. Average DNN confidence for non-pruned DNN and three pruned models with 70%, 80% and 90% pruning respectively. DNN confidence is significantly decreased when applying pruning.

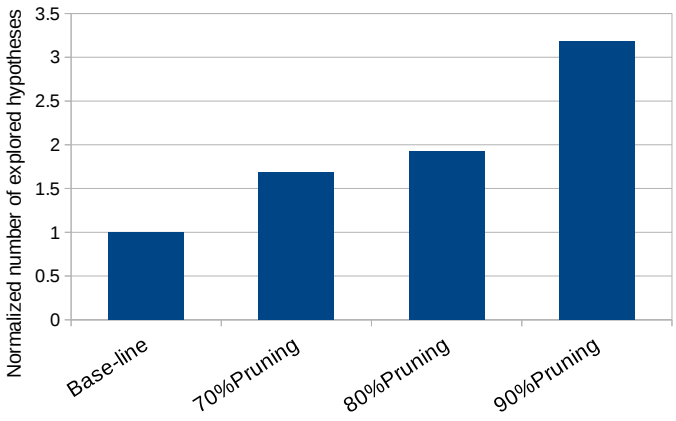

Fig. 4. Normalized number of hypotheses, a.k.a. paths, explored during the Viterbi search for the pruned DNN models at 70%, 80% and 90% of pruning. DNN pruning has a high impact on the workload of the beam search.

a slowdown of 4.5x in the Viterbi beam search, and an overall slowdown of 33% in the ASR system.

C. Impact of DNN Pruning in ASR

The state-of-the-art solution in ASR consists of combining a DNN, to generate acoustic scores, with a Viterbi beam search to find the most likely sequence of words. For example, Microsft's ASR system [12], which is claimed to achieve human parity in speech recognition, employs DNN-computed scores to drive a Viterbi search [13]. Recent works on hardware-accelerated ASR also integrates a DNN with a Viterbi search [14], [15], [16], [17]. On the other hand, solutions based on Recurrent Neural Networks (RNNs) also include a Viterbi beam search. For example, Baidu's Deep-Speech [18] combines an RNN with the Viterbi search algorithm as described in [19], whereas EESEN [20] employs a Viterbi search driven by the scores computed by an LSTM network [3], [21]. Furthermore, regarding the GMM-based solutions, recent software optimization [22] and hardware acceleration [23] have made the GMM evaluation so efficient, that the Viterbi search becomes the main bottleneck of these systems. However, the combination of a DNN with a Viterbi beam search is pervasive in state-of-the-art ASR, as it has been proven to deliver the highest recognition accuracy.

In this paper, we analyze the impact of DNN pruning on the performance of the Viterbi beam search [24], i.e. the consumer of the DNN scores in ASR systems. The Viterbi search takes as input the DNN's acoustic scores and a graph-based model of the language. In a modern ASR system, the language is represented by using a Weighted Finite State Transducer (WFST) [25]. The WFST is a Mealy finite state machine that encodes a mapping between input and output labels in the edges of a graph. Each edge has also a weight that represents the associated cost for this transition. For ASR, the input labels represent sub-phonemes (output classes of the DNN), whereas the output labels represent the words. The WFST is constructed offline during training from multiple knowledge sources, such as pronunciation and grammar, by using powerful statistical learning techniques. For large vocabulary ASR systems, the resulting WFST contains millions of states and arcs. The Viterbi beam search employs a WFST to find the sequence of output labels, i.e. words, with maximum likelihood for the sequence of input labels, or sub-phonemes, whose associated probabilities are computed by the DNN. In this paper, we focus on describing how the DNN affects the workload of the Viterbi search. A more detailed description of the Viterbi algorithm is provided in [15], [17].

Figure 5 illustrates the behavior of the Viterbi beam search for one frame of speech. The left part of the figure shows both the source states and the new states created during the search for that particular frame. Each one of these states represents a partial path or hypothesis, i.e. an alternative representation of the speech from the beginning of the audio signal to the current frame. Each of the partial hypotheses has an associated likelihood, that is computed based on the DNN scores and the information from the WFST, such as the language model. For numerical stability, all the likelihoods are converted to log-space and the sign is ignored. The positive logarithm of the probability is used as the cost for a given path: the smallest the cost, the highest the likelihood of the hypothesis.

The top-right part of Figure 5 shows the costs of the different hypotheses when using the non-pruned DNN. The cost of a new path is computed as the cost of the source state plus the cost of the sub-phoneme associated with the WFST arc being traversed (S_1 for hypothesis 1, S_2 for 2, etc.) for the given frame of speech. In case the arc corresponds to a cross-word transition, the cost from the language model is also added [17]. The DNN provides the costs, i.e. likelihoods, for the different sub-phonemes S_1, S_2, S_3 and S_4 employed in the current frame. As it can be seen, the non-pruned DNN is very confident in its prediction, and it clearly identifies S_2 as the correct sub-phoneme for the current frame. Therefore, hypotheses that include S_2 in this frame obtain very small cost, whereas the hypotheses using incorrect sub-phonemes get high cost. Once all the new hypotheses are created, the path with the smallest cost is identified (hypothesis 2) and

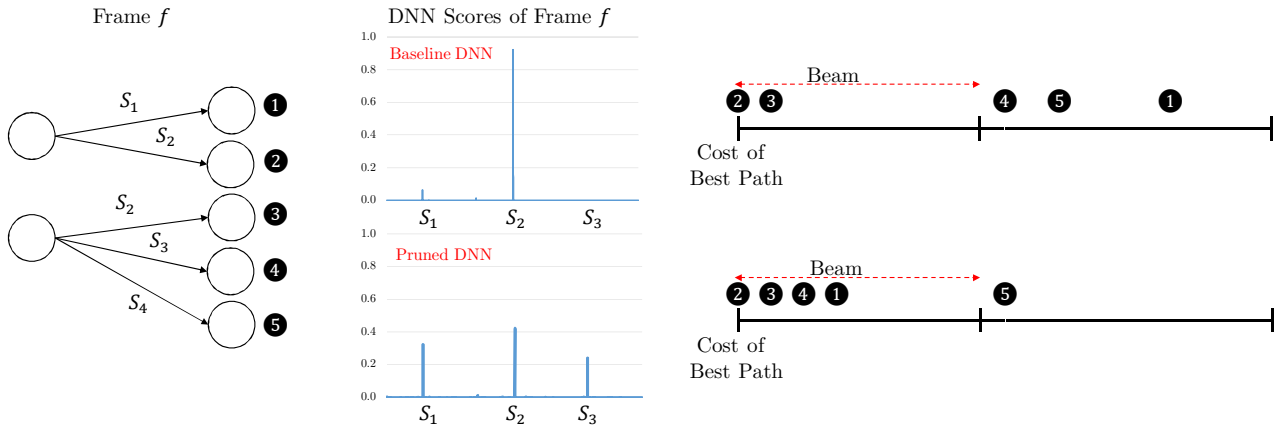

Fig. 5. The figure illustrates the behavior of the Viterbi beam search for one frame of speech, for both the non-pruned DNN (top) and the pruned model (bottom).

hypotheses whose distance to the best path is larger than the beam width are discarded. In the example, hypotheses 4, 5 and 1 are discarded, and only 2 and 3 will be explored in the next frame.

On the other hand, the bottom-right part of Figure 5 shows the costs of the different paths when using the pruned model. The pruned DNN is much less confident in its prediction. Although S_2 is still the most likely sub-phoneme, the DNN assigns a relatively high score to sub-phonemes S_1 and S_3. In this case, hypotheses 1 and 4, that include sub-phonemes S_1 and S_3, obtain a cost much smaller than with the non-pruned model, and fall within the beam distance. Therefore, 1 and 4 are not discarded when using the pruned DNN and, hence, they will be explored in the next frame, increasing the workload of the Viterbi search. This example illustrates how the DNN outputs have a high impact in the costs of the alternative hypotheses and the overall activity for the Viterbi beam search.

Figure 4 shows the increase in Viterbi workload when using the pruned DNN models from Table I. As it can be seen, Viterbi workload increases when the degree of pruning is increased, i.e. when the confidence of the DNN is reduced. Although the 70% of pruning only introduces a confidence loss of 5%, it results in an increase of more than 1.5x in the number of paths explored during the Viterbi search. The confidence of the model pruned at 80% drops by 9%, causing an increase of almost 2x in the Viterbi search workload. Finally, the 90% pruning introduces a confidence loss of 22% in DNN predictions and an increase of more than 3x in the number of hypotheses explored. This increase in activity results in the large slowdowns for the Viterbi search previously reported in Figure 2.

The naive solution for this problem consists of reducing the beam width to discard more hypotheses. This solution does not require changes to software or hardware implementations. However, finding a proper beam width that works for any utterance is complex. If the beam is too small, it may affect Word Error Rate as the correct hypothesis might be early discarded, whereas if the beam is too large the system suffers the aforementioned increase in Viterbi search activity. Furthermore, we have observed that there are some utterances that still suffer the workload increase even with fairly small beams. This results in long tail latencies, that are undesirable for a real-time ASR system. Next section presents a solution to efficiently combine DNN pruning with Viterbi search that does not require the user to change the beam, does not suffer from long tail latencies and requires minor hardware extensions to a Viterbi accelerator.

III. HARDWARE-ACCELERATED ASR

In this section, we present our hardware-accelerated ASR system that manages to efficiently combine DNN pruning with Viterbi beam search. First, we describe the baseline Viterbi search accelerator, focusing on the components that are affected by the workload increase due to the pruning (see Section II-C). Next, we describe how the architecture of the Viterbi accelerator can be extended to mitigate the negative effects of DNN pruning, followed by an analysis of the effectiveness of the hardware extensions. Finally, we describe the DNN accelerator employed in our ASR system, that is able to handle the pruned DNNs.

Our ASR system employs UNFOLD [17] to achieve high-performance and energy-efficient Viterbi search. Figure 6 illustrates UNFOLD's architecture. This accelerator includes several pipeline stages and different on-chip memories. Regarding the pipeline stages, the State and Arc Issuers fetch from memory the information of the WFST states and their outgoing arcs, in order to generate new hypotheses. Next, the Acoustic-likelihood Issuer reads the DNN-computed probabilities for the sub-phonemes associated with the arcs. Then, the Likelihood Evaluation Unit computes the cost of the new hypotheses. Finally, the Hypothesis Issuer stores the new hypotheses' information in a hash table. In this section, we focus on describing the components affected by the DNN pruning, that are later modified in Section III-B to mitigate the negative effects of the pruning. These components are

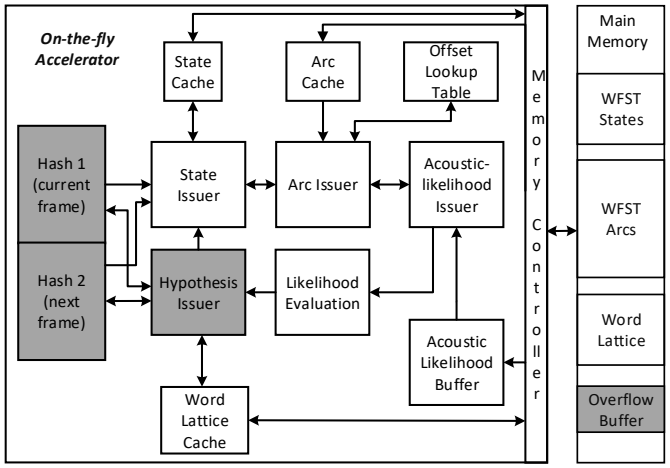

Fig. 6. Architecture of UNFOLD [17], a state-of-the-art Viterbi search accelerator. The main components affected by DNN pruning are highlighted.

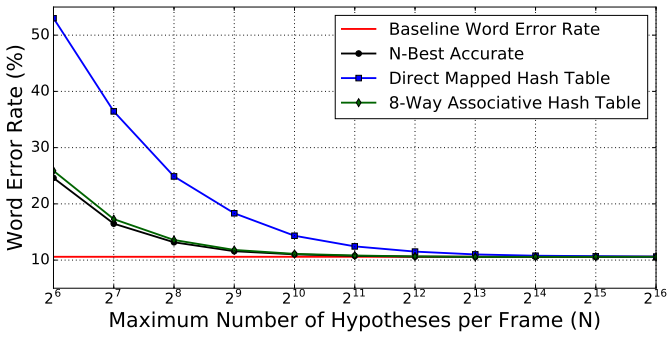

Fig. 7. Word Error Rate (WER) versus maximum number of hypotheses explored per frame (N). An 8-way associative hash table with 1024 entries (128 sets) achieves nearly the same WER than the baseline system that explores an unbounded number of hypotheses. In addition, our 8-way associative hash table exhibits very similar behavior to the system that accurately tracks the N-best hypotheses (N-Best Accurate).

highlighted in Figure 6. A more thorough description of UNFOLD is provided in [17].

A. Baseline Viterbi Search Accelerator

UNFOLD employs two hash tables to store the different hypotheses for the current frame of speech and the next frame. These hash tables are direct mapped. When a new hypothesis is generated, it is inserted in its corresponding entry in the hash table, computed with an XOR hash function of the hypothesis' information. In case of a collision, i.e. in case the corresponding entry is already occupied by a different hypothesis, the hash table provides a backup buffer. All the hypotheses that caused a collision are then stored in the backup buffer, and hypotheses that correspond to the same entry are linked. Therefore, accesing the hash table may take multiple cycles in case of a collision, whereas it only requires one cycle for accessing the direct mapped region. The number of hypotheses for a given frame of speech may exceed the on-chip resources, in this case UNFOLD employs an overflow buffer in system memory to store the hypotheses, introducing large delays due to main memory latency.

We have measured the number of hypotheses per frame for LibriSpeech [11] test set audio files, and found that the average number of hypotheses per frame is larger than 20K, whereas the maximum number is bigger than 300K. UNFOLD's hash table [17] contains 32K entries in the direct mapped region and 16K entries in the backup buffer. This configuration avoids collisions and overflows by a large extent when using the non-pruned DNN, i.e. most of the time only the direct mapped region is used and, hence, most of the accesses are served in one cycle. However, when using the pruned DNN, the number of hypotheses increases significantly as shown in Figure 4, producing a large increase in the number of collisions and overflows of on-chip resources. Due to the latency to access main memory, overflows have a huge impact in the performance of the accelerator. For example, the pruned DNN at 90% produces an increase in the number of hypotheses of 3.1x, causing an slowdown of 4.5x in UNFOLD. Next section presents a novel hash table design that guarantees single-cycle latency for accessing the hypotheses, even when using the pruned DNNs.

B. Hash Table Design for N-Best Selection

The key idea in our design is to constrain the Viterbi search to the N-best hypotheses, i.e. the N paths with the smallest cost, on every frame of speech. Therefore, only up to N hypotheses have to be stored per frame, independent to the confidence of the DNN scores, avoiding the explosion in Viterbi search workload. If N is small enough, all the hypotheses can be stored on-chip, avoiding accesses to the overflow buffer in main memory. Note that states in the WFST have multiple outgoing arcs (see left part of Figure 5) and, hence, exploring the N-best paths from the previous frame results in M hypotheses, where M is bigger than N (typically between 2x and 4x bigger in our experiments). Accurately selecting the N paths with the smallest cost out of the M generated hypotheses requires a partial sort. This sort is computationally expensive, since the values of N and M should be in the order of thousands to maintain high recognition accuracy.

In this work, we take a different approach and introduce a novel design that loosely selects the N-best hypotheses by using a K-way set-associative hash table. Our solution maintains the K-best hypotheses mapped to each set. In case more than K paths are mapped to the same set, our system keeps the K paths with the smallest cost and discards the rest. By doing so, no access to the backup buffer or overflow buffer is required. However, our system may potentially discard hypotheses that are among the best N, since more than K paths from the top-N may be mapped to the same set.

Figure 7 shows that our system exhibits very similar behavior to the one that accurately selects the N-best paths. The red line shows the Word Error Rate (WER) for the baseline system that explores an unbounded number of hypotheses (10.59% for the 5.4 hours of speech from LibriSpeech test

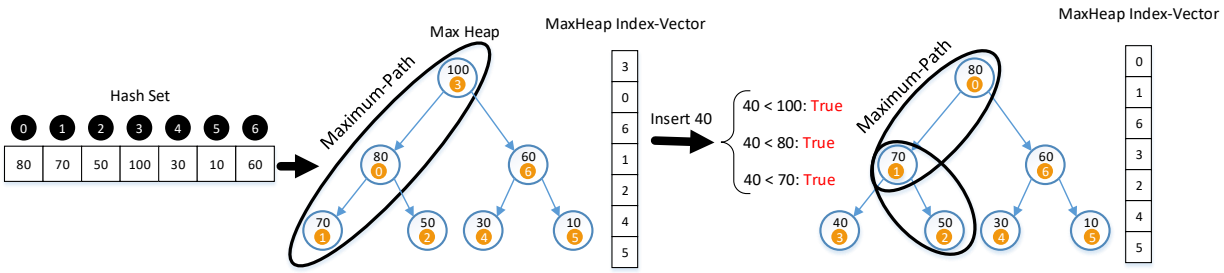

Fig. 8. An example of the Max-Heap binary tree for a set with 7 hypotheses, that illustrates how the replacement of the worst hypothesis is performed in our hash table.

set). The other three lines show the WER for accurate N-best selection and our system using a direct mapped and an 8-way associative hash table. The x-axis shows the value of N, i.e. the maximum number of hypotheses kept per frame. As it can be seen, an 8-way associtive hash table provides very similar WER to the system that accurately selects the top-N hypotheses for different values of N. In addition, a hash table with just 1024 entries achieves a WER of 11%, extremely similar to the baseline UNFOLD accelerator (10.59%). Note that associativity has a high impact on our solution, since the direct-mapped hash table drops significant accuracy with less than 4096 entries.

Therefore, using a small 8-way associative hash table avoids the workload increase in the Viterbi search due to the DNN pruning, since the maximum number of hypotheses per frame is bound, while requiring simpler hardware (backup buffer and overflow buffer are not required) and achieving high recognition accuracy. However, the issue with our approach is the replacement policy in the sets of the hash table. In case a new hypothesis is inserted in a full set, the hypothesis with maximum cost within the set must be replaced. For a set with 8 entries, a three-level tree of comparators can be used to locate the entry with maximum cost, this solution includes three sequential comparisons in the critical path. We synthesized this circuit and obtained a critical path delay of 2.82 ns, which requires 3 cycles in the UNFOLD accelerator (1.25 ns cycle time). Note that the target of our hash design is to deliver single-cycle latency for accessing the hash table, as higher latencies may introduce stalls in the pipeline of the accelerator (see Section III-A).

In order to efficiently handle the replacements in the sets of the hash table, we employ a Max-Heap on each set. The Max-Heap provides fast access to the entry with maximum cost, i.e. the hypothesis that must be replaced. In addition, we show that by using the Max-Heap, all the comparisons required to do the replacement and update the heap can be performed in parallel, achieving single-cycle latency for accessing the hash table. Figure 8 shows an example of a Max-Heap generated after the insertion of 7 hypotheses into a set. For the sake of simplicity, we consider seven as the size of the set. As it can be seen, this structure assures that each node at all the levels of tree has higher value than its descendants. Then, for a replacement in a set of the hash table, we only have to compare the new hypothesis' cost with the root node, i.e. the node with the maximum cost of the set, and replace it if the cost of the new hypothesis is lower.

As depicted in Figure 8, we use a Max-Heap index vector at each set that stores the indices of the hypotheses at the different nodes of the Max-Heap tree. In general, the positions of the children of a node at index n of this table are located at indices $2n+1$ and $2n+2$. When replacing a hypothesis in a set by removing the Max-Heap's root, the new hypothesis may be placed at different levels in order to recover the Heap condition. These locations are specified by a Maximum-path as shown in Figure 8. The Maximum-path is the path including the successors with maximum cost starting from the root node to a leaf node, and it is stored as meta-data of the set and updated on every insertion. Figure 8 shows an example of inserting a hypothesis with cost 40. By comparing this cost with the cost of Maximum-path's nodes, we can find the location of the new hypothesis. Note that all the comparisons can be done in parallel. As the new hypothesis' cost is lower than all the nodes, it should be placed at the last level of the Max-Heap by shifting up nodes with cost 70 and 80. Accordingly, the Max-Heap Index-vector is updated to show the changes made to the Max-Heap tree. Moreover, the Maximum-path is updated, as it will be required for future insertions.

As described in the previous paragraph, our design is able to efficiently replace a hypothesis and update the Max Heap tree, using several parallel comparators and two index vectors storing the order of the set's indices in the Max-Heap and the Maximum-path respectively. Unlike the tree of comparators, our solution performs all the comparisons in parallel. By using the outcome of these comparsions and the aforementioned index vectors, reordering the tree due to a replacement in the Heap is performed as cheaply as moving the 3-bit indicies instead of moving the whole information of each set's entry. Note that the data of the entries is not moved or shifted, only the Max-Heap Index-Vector, that contains 3 bits per entry, is updated on an insertion. To verify that all the required operations for a replacement finish at one cycle, we modeled our design in Verilog and synthesized it using the Synopsys Design Compiler tool. Our synthesis results show a critical path delay of 1.21 nano-seconds, which is lower than the cycle time in UNFOLD (1.25 ns).

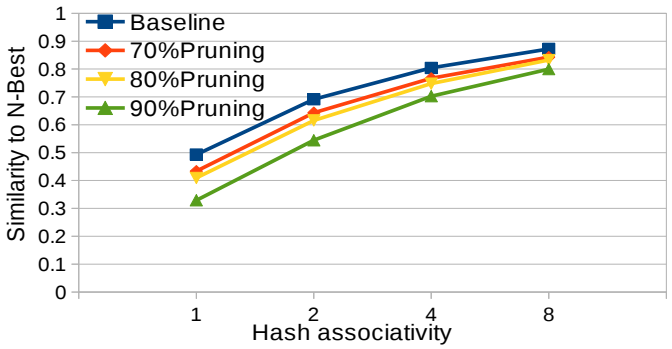

Fig. 9. Similarity between a system that accurately selects the N best hypotheses and our system that loosely tracks the N best paths, for different degrees of pruning and associativities.

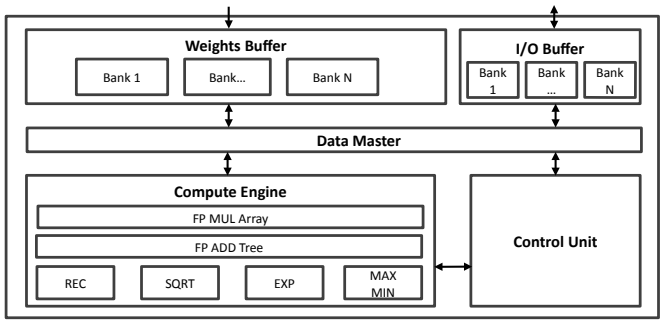

Fig. 10. Architecture of the DNN accelerator used in our ASR system.

In short, our approach simpilifies UNFOLD's architecture by removing the backup buffer and the overflow mechanism. In addition, our hash table only requires 1024 entries, whereas UNFOLD's hash table has 32K entries. As a result, the overal area of the accelerator is reduced by 2x. Furthermore, by adding our proposed replacement mechanism, we provide a predictable access time of one cycle for the hash table architecture. The replacement scheme represents a negligible overhead of 6% and 0.2% increase in the total area and power-dissipation respectively.

C. Analysis of Hash-Based N-Best Approach

We have analyzed the accuracy of our novel hash table design in selecting the N most promising hypotheses. We compare our proposal with a system that accurately selects the N-best hypotheses in each frame of speech. Figure 9 shows the similarity measured as the number of hypotheses chosen in both systems divided by N. As it can be seen, the bigger the associativity the higher the accuracy of our proposal in selecting the N-best paths. An 8-way hash table achieves a similarity between 80% and 90% for different DNN models. On the other hand, as the degree of pruning is increased the similarity is reduced. More aggressive pruning produces an increase in the number of hypotheses, which causes more replacements in our hash table, potentially discarding more paths that are among the N-best hypotheses.

D. DNN Accelerator Overview

In this subsection, we describe the DNN accelerator employed in our ASR system. This DNN accelerator is loosely based on DaDianNao [26], but we have extended it to support pruned models. A high-level block diagram of the accelerator is provided in Figure 10. The Compute Engine (CE) contains the functional units that perform the FP computations, including an array of FP multipliers and a tree of FP adders, together with specialized functional units (reciprocal, square root...). On the other hand, the Control Unit (CU) provides the appropriate control signals in every cycle. The CU contains the configuration of the DNN.

Regarding the on-chip storage, the accelerator includes an eDRAM memory to store the non pruned synaptic weights of the different layers, and the indices that indicate the correspondence between weights and inputs. eDRAM is used in our design to provide larger on-chip capacity with a small cost in area compared to SRAM. This memory is highly multi-banked to achieve the required bandwidth to feed a large number of functional units in the CE.

On the other hand, the I/O Buffer is an SRAM memory used to store the intermediate inputs/outputs between two DNN layers. This memory is highly multi-banked and multi-ported to be able to obtain enough inputs per cycle while avoiding bank conflicts by a large extent. Finally, the Data Master is in charge of fetching the corresponding weights and inputs from the on-chip memories, and dispatching them to the functional units in the CE.

The accelerator is configured for different DNNs by loading the neural network model that includes the information of each layer, i.e. number of input neurons, number of output neurons, weights, biases and indices. Since for our DNNs all the parameters fit in the on-chip memories of the accelerator, the entire dataset is loaded from main memory. By doing so, weights stored on-chip are reused across multiple DNN executions and, hence, no additional off-chip memory access is required for those elements. The accelerator is power gated during idle periods and, hence, weights are loaded from main memory at the beginning of processing every sequence of inputs (i.e. audio utterance).

The vast majority of the computations for MLPs come from FC layers. The FC layer computes the dot product of the inputs and the weights of each neuron. The weights for an FC layer are stored in order in the Weights Buffer. That is, the weights of the first neuron plus its indices are stored first, then the second neuron and so on. The inputs are interleaved within multiple banks in the I/O Buffer so that multiple inputs can be read at the same time from different banks. The main challenge for supporting the pruned models is fetching the inputs, since a group of M weights from a given neuron requires a sparse set of M inputs. On each cycle, M different and non-consecutive (due to the pruning) inputs have to be fetched from an I/O Buffer with B banks and P ports per bank. If the number of collisions in the worst case, i.e. the number of different indices mapped to the same bank, is smaller or equal than P, the I/O Buffer provides all the inputs in one cycle. Otherwise,

fetching the inputs takes multiple cycles, introducing stalls in the pipeline of the accelerator.

The execution of an FC layer works as follows: initially, the accelerator reads the first M weights of the same output neuron and their corresponding indices. Then, the indices are used to fetch the inputs from the I/O Buffer. If the inputs required are located in different banks or there are less or equal than P indices mapped to the same bank (being P the number of read ports per bank), the I/O Buffer provides all the inputs in one cycle. Otherwise, the accelerator will stall until obtaining all the required inputs. When all the inputs are available the accelerator performs M MULs of the inputs by the weights, followed by a reduction using the tree adder to accumulate the result of the neuron. In parallel, the accelerator will start reading the next M weights and indices, which can be from the same neuron if not finished yet or the next one. The neurons's output is finally stored in the I/O Buffer, to be used by the next layer. This process is repeated until all the neurons have been evaluated. The accelerator is pipelined, so if there are no conflicts, in the same cycle the I/O Buffer reads M inputs, the Data Master reads M weights from memory and the CE performs M multiplications and M additions.

Regarding the efficiency of the accelerator's compute engine, we measured the FP throughput drop for the pruned models at 70%, 80% and 90% of pruning to be 11%, 18% and 33% respectively, for the DNN accelerator with parameters provided in Section IV. In other words, for 90% of pruning the utilization of the FP units drops by 33% with respect to the non-pruned DNN. This reduction in FP throughput is due to the sparsity of the pruned models, which generates conflicts in the I/O Buffer. Despite these conflicts, pruning still provides significant speedups and energy savings as reported in Section V. Although the processing of the remaining connections after the pruning is less efficient, pruning removes a large percentage of the weights, reducing computations and memory accesses by a large extent.

Multiple instances of the accelerator shown in Figure 10 can be integrated in the same chip to improve performance and accommodate large DNNs. Each instance of the accelerator, or tile, includes a router to communicate results with the other tiles. The different tiles are connected in a ring. For FC layers, output neurons are evenly distributed among the tiles.

IV. Evaluation Methodology

In order to evaluate our hardware-accelerated ASR system, we developed two separate simulators that accurately model the Viterbi accelerator described in Section III-A and the DNN accelerator for pruned DNNs explained in Section III-D. Furthermore, we have implemented in the Viterbi simulator all the hardware extensions for the new hash table as described in Section III-B. The configuration parameters used in each accelerator are shown in Table II and Table III. For the Viterbi accelerator, most of the parameters are taken from [17], whereas for the DNN accelerator we performed a design space exploration to select the configuration which provides the best trade-off considering performance, area and energy consumption.

TABLE II
Parameters for the DNN accelerator.

Number of Tiles	4
Number of 32-bit multipliers	128
Number of 32-bit adders	128
Weights Buffer	18 MB
I/O Buffer	32KB, 64 Banks - 2RD and 1WR ports

TABLE III
Parameters for the Viterbi accelerator.

State Cache	256 KB, 4-way, 64 B/line
Arc Cache	768 KB, 8-way, 64 B/line
Word Lattice Cache	128 KB, 2-way, 64 B/line
Acoustic Likelihood Buffer	64 Kbytes
Hash Table	100KB, 6 FP comparators
Memory Controller	32 in-flight requests
Likelihood Evaluation Unit	4 FP adders, 2 FP comparators

Regarding the DNN model, we use an MLP for acoustic scoring implemented in the Kaldi [27] toolkit, a popular framework for speech recognition, trained with LibriSpeech [28] dataset. The DNN accelerator has four tiles, with a total of 128 FP adders and 128 FP multipliers (32 per tile). It includes 18MB of eDRAM for the Weights Buffer (4.5 MB per tile), which is enough to fit on-chip *Kaldi*'s baseline DNN model. In the case of the pruned models, the eDRAM banks that are not used are power gated to reduce energy consumption. For instance, the 70% pruning only requires 6.7 MB, the 80% 4.4 MB and the 90% 2.2 MB. Note that the total size of the pruned model also accounts for the indices of the weights, required to locate the inputs, and the biases of non-pruned neurons. Finally, the I/O Buffer is sized to fit the inputs and outputs of the DNN, which requires 32KB, and it is multi-banked with a total of 64 banks with two read ports each, to be able to provide the 128 elements required per cycle if there are no conflicts.

All the Viterbi accelerator's pipeline stages and the DNN's combinational logic components are implemented in Verilog and synthesized to obtain the delay and power using the Synopsys Design Compiler, the modules of the DesignWare library and the technology library of 28/32nm from Synopsys [29]. On the other hand, we characterize the memory components of the accelerators by obtaining the delay, energy per access and area using CACTI-P [30]. For both the Design Compiler and CACTI, we use the technology library and the configurations optimized for low power and a supply voltage of 0.78 V. Finally, the energy consumption of main memory is estimated by using MICRON's power model for LPDDR4 [31]. The simulators provide the activity factors for the different components and the total cycle count, which are then used to compute execution time, and dynamic and static energy by combining them with the estimations of the Design Compiler, CACTI and MICRON's power models.

To set the frequency of the system, we consider the critical path delay and access time reported by Design Compiler and

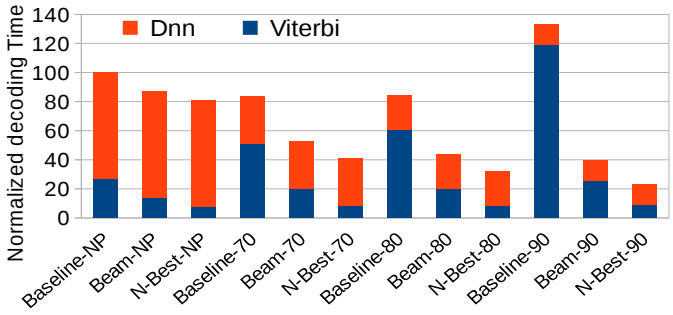

Fig. 11. Execution time for the entire ASR system, including the breakdown between execution time of DNN and Viterbi accelerators. Time is normalized to the configuration *Baseline-NP*, i.e. the baseline ASR system with the non-pruned DNN.

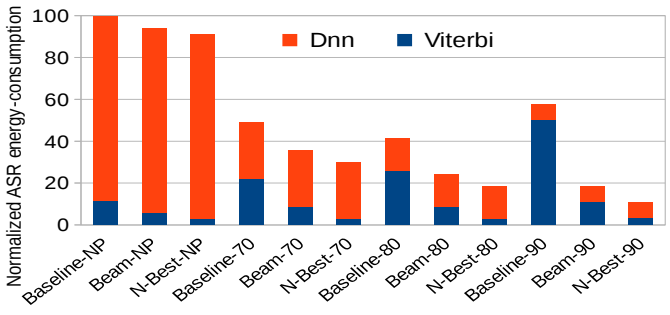

Fig. 12. Normalized energy for the entire ASR system, including the breakdown between energy consumption of the DNN and Viterbi accelerators. Baseline configuration is labeled as *Baseline-NP*, i.e. the baseline ASR system with the non-pruned DNN. Numbers include both static and dynamic energy.

CACTI respectively. We take the maximum delay among the different components, which is 1.25 ns and 2 ns for the DNN and Viterbi accelerators, resulting in 800 MHz and 500 MHz respectively.

The integration of the two accelerators is performed as described in [16]. In our ASR system, the DNN and Viterbi accelerators work independently and communicate through a shared buffer in system memory. The DNN accelerator computes the acoustic scores for the different frames of the speech signal and it stores them in the shared buffer. Then, the Viterbi accelerator performs the search for each frame by fetching from main memory the acoustic scores computed by the DNN accelerator. Our simulations account for the time and energy required by the accesses to this shared buffer.

V. Experimental Results

This section presents the experimental results achieved by our ASR system that efficiently combines the DNN prunning with the Viterbi beam search. The baseline configuration is a hardware-based ASR system that includes two accelerators: the DNN accelerator optimized for pruned fully-connected networks described in Section III-D, and the state-of-the-art Viterbi search accelerator presented in [17]. The baseline system is able to employ both the non-pruned DNN and the pruned models. We label these configurations as *Baseline-NP*, *Baseline-70*, *Baseline-80* and *Baseline-90* for the models at 0% (non-pruned), 70%, 80% and 90% of pruning.

In order to deal with the increase in Viterbi search workload due to the DNN pruning (see Section II), we test two different solutions. The first one consists on reducing the beam width while keeping the hardware unmodified. When using the pruned DNNs, we reduce the beam to the minimum value that is able to retain Word Error Rate. We use a beam width, in log-space, of 15, 10, 9 and 8 when using the pruned DNN at 0%, 70%, 80% and 90% of pruning respectively. We label these configurations as *Beam-NP* (non-pruned), *Beam-70*, *Beam-80* and *Beam-90*. The second solution consists on extending the Viterbi search accelerator to loosely keep track of the N-best hypotheses, as described in Section III. We use 1024 as the value of N. We label these configurations as *NBest-NP* (non-pruned), *NBest-70*, *NBest-80* and *NBest-90*.

Figure 11 shows the normalized execution time for the different configurations, including the breakdown between execution time for the DNN accelerator and the Viterbi search accelerator. The results are normalized to the execution time of the *Baseline-NP* configuration, i.e. the baseline hardware-accelerated system with the non-pruned DNN. Regarding the performance of the DNN accelerator, pruning provides substantial speedups as expected. Pruning improves the performance of the DNN accelerator by 2.3x, 3.1x and 5.1x for degrees of pruning of 70%, 80% and 90% respectively. Note that *Baseline*, *Beam* and *NBest* employ the same DNN accelerator, they only differ in the implementation of the beam search. On the other hand, pruning introduces large slowdowns for Viterbi in the *Baseline* ASR system, producing an increase in execution time of the beam search of 1.9x, 2.3x and 4.5x for 70%, 80% and 90% of pruning respectively. Note that the performance loss in Viterbi search offsets part of the benefits from DNN pruning for *Baseline-70* and *Baseline-80*, and it produces 33% slowdown for the entire ASR system for *Baseline-90*. This slowdown is caused by the workload increase due to the reduction in DNN confidence, as explained in Section II-C and Figure 4.

Reducing the beam width in the *Beam* configurations avoids the workload explosion in the Viterbi search, as more hypotheses are discarded reducing the search space. *Beam-NP* obtains 12.7% speedup with respect to *Baseline-NP*, as we found that we could slightly reduce the beam with respect to the default setup in Kaldi without affecting Word Error Rate. *Beam-70*, *Beam-80* and *Beam-90* reduce the execution time of the overall ASR system by 47.5%, 56.5% and 60% respectively. However, Viterbi search execution time is still increased when pruning: *Beam-90* exhibits an slowdown of 1.8x in the Viterbi beam search with respect to *Beam-NP*. Although the smaller beam prevents the workload increase in the Viterbi for many utterances, we found that some audio files still suffer the explosion of activity in the beam search, causing long tail latencies.

On the other hand, our *NBest* configuration solves the problem of the workload increase in Viterbi by loosely tracking the best 1024 paths on every frame of speech, establishing

an upper bound for the complexity of the beam search since only 1024 paths will be kept per frame, independently of the confidence of the DNN. For the non-pruned DNN, *NBest-NP* provides 3.5x speedup for Viterbi search and 20% reduction in the execution time of the overall ASR system with respect to the *Baseline-NP*. Unlike *Baseline* and *Beam* configurations, our *NBest* system does not suffer any slowdown in the Viterbi beam search when applying the pruning. *NBest-90* achieves 4.2x speedup with respect to *Baseline-NP*. In addition, it improves the performance of the *Beam-90* by 1.69x. Therefore, our *NBest* solution is able to effectively combine both DNN pruning and the beam search, providing an implementation where the complexity of the search is not affected by the confidence of the DNN predictions.

Figure 12 shows the normalized energy for the same configurations, including both static and dynamic energy. Regarding the DNN accelerator, pruning reduces energy consumption by 3.3x, 5.7x and 11.8x for 70%, 80% and 90% of pruning respectively. These savings come from the reduction in activity in the accelerator due to the pruning, as the pruned models require significantly less computations and memory accesses. Regarding the Viterbi accelerator, pruning increases its energy consumption in the *Baseline* system by up to 4.3x (90% of pruning). This increase in energy consumption is due to the large workload increase due to the reduction in DNN confidence. The Viterbi accelerator explores more paths when using the pruned models, as previously reported in Figure 4, which requires more memory accesses and computations. The *Beam-90* configuration suffers an increase in Viterbi search energy consumption of 1.8x with respect to *Beam-NP*. On the other hand, our *NBest* solution is able to maintain the energy consumption of the beam search when applying pruning.

The *NBest-90* provides 5.25x energy reduction for the overall ASR system with respect to the *Baseline-90* configuration, and 1.67x energy reduction with respect to *Beam-90*. The energy savings come from multiple sources. First, dynamic energy is reduced since our system reduces the number of hypotheses explored, as described in Section III. Therefore, it requires significantly less memory accesses and computations due to the smaller search space. Second, static energy is reduced due to the speedups reported in Figure 11. In addition, the area of our accelerator is reduced since the hash table employed to maintain the hypotheses is smaller (see Table III). Our *NBest* system only has to maintain up to 1024 alternative paths in the worst case, whereas *Baseline* and *Beam* have to provide hardware resources for tens of thousands of alternative paths. To this end, UNFOLD [17] provides a hash table with tens of thousands of entries, an on-chip backup buffer to efficiently handle collisions in the hash table, and an overflow buffer in system memory to be used in case the number of alternative hypotheses exceeds the on-chip resources. In comparison, our system only provides a much smaller hash table, in case of collisions in a set the path with the worst cost is discarded. This results in a reduction in area from 21.45 mm^2 (UNFOLD) to 10.74 mm^2.

In short, our ASR system based on loosely tracking the N-best hypotheses on each frame of speech is able to efficiently integrate DNN pruning with Viterbi search. In our system, the DNN pruning improves the performance and energy consumption of the DNN accelerator without introducing any penalty in the Viterbi search accelerator. Our solution provides 4.2x speedup and 9x energy savings with respect to the state-of-the-art in ASR (*NBest-90* versus *Baseline-NP*). Compared to a system that directly applies DNN pruning without taking any action to prevent the workload increase in the Viterbi (*Baseline-90*), our ASR system (*NBest-90*) delivers 5.65x speedup and 5.25x energy savings. Finally, compared to a system that reduces the beam width to mitigate the impact of the loss in DNN confidence (*Beam-90*), our system improves performance and energy consumption by 1.69x and 1.67x respectively.

VI. Related Work

This section outlines previous work on hardware acceleration for DNNs and Viterbi search, especially in the context of ASR systems.

Viterbi Accelerators. Accelerating the Viterbi search in ASR systems has attracted the attention of the architectural community in recent years. Price et al. [14], [32], [33] propose a low-power speech recognition system that integrates a DNN accelerator for non-pruned models with a Viterbi search accelerator. On the other hand, Reza et al. [15], [16], [17] present an ASR system that combines a GPU for evaluating non-pruned DNNs with a high-performance and energy-efficient Viterbi search accelerator. Our work is different from previous proposals since we include a DNN accelerator for pruned models. Furthermore, we extend the state-of-the-art Viterbi accelerator presented in [17] to loosely track the N-best hypotheses on each frame of speech, avoiding the workload explosion in Viterbi search due to the loss of confidence in the DNN. To the best of our knowledge, this is the first work that analyzes the interaction between DNN pruning and Viterbi beam search, proposing a novel solution to mitigate the negative effects of the pruning in an ASR system.

DNN Accelerators. A reference work for DNN accelerators is the DianNao [34], and their variations DaDianNao [26] for large scale and ShiDianNao [35] for mobile devices. DianNao was the first accelerator proposal that includes its own on-chip SRAM buffers to reduce the memory accesses, and DaDianNao further improved this aspect by adding eDRAM to store the weights. Most of the accelerators presented in recent years are based on DianNao, such as Minerva [36]. The most recent works on accelerators are focused on low power [4], [36], [35]. A common technique to reduce power is the pruning [1], [2], [3], [4], [5], [6] of the neural network to reduce the amount of computations and consequently the energy consumption. The result of the pruning is a sparse model which may introduce inefficiencies in a DNN accelerator due to its irregularity. The DNN accelerator employed in our ASR system includes the required hardware to support pruned models. Our work is different from previous accelerators as we apply the pruning

to the fully-connected layers of an MLP, while previous works focused on sparse convolutional neural networks [5].

VII. Conclusions

This work analyzes the impact of DNN pruning in the performance and energy consumption of an Automatic Speech Recognition (ASR) system. Although the pruned DNNs are able to maintain accuracy, the confidence of the DNN predictions, i.e. the likelihood assigned to the top-1 class, is reduced by more than 20%. This confidence loss results in a large increase in the workload of the Viterbi beam search, the consumer of the DNN scores in an ASR system. Since the pruned models are not able to clearly identify the correct sub-phoneme for each frame of speech, many more alternative hypotheses are explored during the Viterbi search, producing a slowdown of 33% for the overall ASR system. In order to mitigate the negative effects of DNN pruning, we propose to extend the hardware of a state-of-the-art Viterbi search accelerator to loosely keep track of the N-best hypotheses on each frame of speech. Constraining the search to N hypotheses per frame avoids the workload explosion due to the confidence loss in DNN predictions, while maintaining Word Error Rate by exploring the most promising hypotheses. Our ASR system manages to effectively combine DNN pruning with Viterbi search. It achieves 4.2x speedup and 9x energy savings with respect to the state-of-the-art ASR solutions.

Acknowledgment

This work was supported by the Spanish State Research Agency under grant TIN2016-75344-R (AEI/FEDER, EU).

References

[1] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proceedings of NIPS*, 2015, pp. 1135–1143.

[2] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[3] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang, and W. J. Dally, "Ese: Efficient speech recognition engine with sparse lstm on fpga." in *FPGA*, 2017, pp. 75–84.

[4] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *Proceedings of ISCA*, 2016, pp. 243–254.

[5] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *Proceedings of ISCA*, 2017, pp. 27–40.

[6] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," in *Proceedings of ISCA*, 2017, pp. 548–560.

[7] X. Zhang, J. Trmal, D. Povey, and S. Khudanpur, "Improving deep neural network acoustic models using generalized maxout networks," in *Proceedings of Acoustics, Speech and Signal Processing*, 2014, pp. 215–219.

[8] M. Denil, B. Shakibi, L. Dinh, N. de Freitas *et al.*, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems*, 2013, pp. 2148–2156.

[9] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The kaldi speech recognition toolkit," in *IEEE Workshop on Automatic Speech Recognition and Understanding*, 2011.

[10] R. Haeb-Umbach and H. Ney, "Linear discriminant analysis for improved large vocabulary continuous speech recognition," in *Proceedings of Acoustics, Speech, and Signal Processing*, 1992, pp. 13–16.

[11] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An asr corpus based on public domain audio books," in *Proceedings Acoustics, Speech and Signal Processing*, 2015, pp. 5206–5210.

[12] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig, "Achieving human parity in conversational speech recognition," *CoRR*, 2016.

[13] C. Mendis, J. Droppo, S. Maleki, M. Musuvathi, T. Mytkowicz, and G. Zweig, "Parallelizing wfst speech decoders," in *Proceedings of Acoustics, Speech and Signal Processing*, 2016, pp. 5325–5329.

[14] M. Price, "Energy-scalable speech recognition circuits (doctoral dissertation)," in *Massachusetts Institute of Technology*, 2016. [Online]. Available: http://hdl.handle.net/1721.1/106090

[15] R. Yazdani, A. Segura, J. M. Arnau, and A. Gonzalez, "An ultra low-power hardware accelerator for automatic speech recognition," in *Proceedings of MICRO*, 2016, pp. 1–12.

[16] R. Yazdani, A. Segura, J.-M. Arnau, and A. Gonzalez, "Low-power automatic speech recognition through a mobile gpu and a viterbi accelerator," *IEEE Micro*, vol. 37, no. 1, pp. 22–29, 2017.

[17] R. Yazdani, J. M. Arnau, and A. González, "Unfold: A memory-efficient speech recognizer using on-the-fly wfst composition," in *Proceedings of MICRO*, 2017, pp. 69–81.

[18] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates *et al.*, "Deep speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[19] A. Y. Hannun, A. L. Maas, D. Jurafsky, and A. Y. Ng, "First-pass large vocabulary continuous speech recognition using bi-directional recurrent dnns," *arXiv preprint arXiv:1408.2873*, 2014.

[20] Y. Miao, M. Gowayyed, and F. Metze, "Eesen: End-to-end speech recognition using deep rnn models and wfst-based decoding," in *Workshop on Automatic Speech Recognition and Understanding*, 2015, pp. 167–174.

[21] F. Silfa, G. Dot, J. Arnau, and A. González, "E-PUR: an energy-efficient processing unit for recurrent neural networks," *CoRR*, vol. abs/1711.07480, 2017.

[22] H. Tabani, J. M. Arnau, J. Tubella, and A. Gonzalez, "Performance analysis and optimization of automatic speech recognition," *Multi-Scale Computing Systems, IEEE Transactions on*, 2017.

[23] H. Tabani, J.-M. Arnau, J. Tubella, and A. Gonzalez, "An ultra low-power hardware accelerator for acoustic scoring in speech recognition," in *Proceedings of Parallel Architecture and Compilation Techniques*, 2017.

[24] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, Feb 1989.

[25] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech and Language*, vol. 16, no. 1, pp. 69 – 88, 2002.

[26] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," in *IEEE 47th MICRO*, 2014.

[27] D. Povey, "Kaldi software," http://kaldi-asr.org/, accessed: 2017-07-20.

[28] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *IEEE ICASSP*, 2015.

[29] "Synopsys," https://www.synopsys.com/, accessed: 2017-07-20.

[30] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-p: Architecture-level modeling for sram-based structures with advanced leakage reduction techniques," in *IEEE/ACM ICCAD*, 2011.

[31] "Micron lpddr4 system power calculator," https://www.micron.com/support/tools-and-utilities/power-calc, accessed: 2017-07-20.

[32] M. Price, J. Glass, and A. P. Chandrakasan, "A 6 mw, 5,000-word real-time speech recognizer using wfst models," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 102–112, Jan 2015.

[33] M. Price, A. Chandrakasan, and J. R. Glass, "Memory-efficient modeling and search techniques for hardware asr decoders," in *INTERSPEECH*, 2016, pp. 1893–1897.

[34] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ACM ASPLOS, 2014.

[35] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *ACM/IEEE 42nd ISCA*, 2015.

[36] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *IEEE 43rd ISCA*, 2016.