UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Ultra low-power, high performance accelerator for speech recognition

Reza Yazdani Aminabadi

Ultra Low-Power, High Performance Accelerator for Speech Recognition

Reza Yazdani Aminabadi

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Doctor of Philosophy

Department of Computer Architecture
Universitat Politècnica de Catalunya

Advisors: Jose-Maria Arnau, Antonio González

May, 2019
Barcelona, Spain

Abstract

Automatic Speech Recognition (ASR) is undoubtedly one of the most important and interesting applications in the cutting-edge era of Deep-learning deployment. ASR technology has introduced a new machine-human interaction paradigm for most of today's mobile systems and wearables. Smart phones and smart watches are the two tangible examples which use ASR as one their main interaction tools. Despite the increasing popularity of ASR, running it on small-form battery-operated devices comes with several main challenges: delivering high accuracy and real-time performance with a small memory and energy budget. Complex ASR systems require high power to achieve real-time, hence, they cannot be simply deployed on ultra low-power systems. The objective of this thesis is to address these issues and propose several novel solutions in order to design a highly efficient customized acceleration architecture for ASR systems, by relaxing the main bottlenecks of the huge memory and computation requirements.

As a first step, we characterize the ASR application running on General-Purpose-Architectures (GPAs) such as CPU and GPU. A state-of-the-art ASR system consists of two major components: acoustic-scoring using DNN and speech-graph decoding using Viterbi search. We observe that Viterbi search is the main performance bottleneck. Thus, we develop an architecture for offloading the Viterbi search to an accelerator. Afterwards, we study the search behavior on our design, pinpointing the main sources of pipeline stalls. After identifying the memory subsystem as the main bottleneck, we introduce two optimization techniques: a prefetching scheme tailored to the needs of the ASR system; and a novel bandwidth saving technique that removes 20% of the off-chip memory accesses. The end design achieves 1.7x speedup over a highly optimized CUDA implementation running on a high-end Geforce GTX 980 GPU, while reducing by two orders of magnitude (287x) the energy-consumption.

Accurate, real-time ASR requires huge memory storage and computational power. State-of-the-art ASR systems run Viterbi search on a vast Weighted Finite State Transducer (WFST). The WFST is a graph-based model created by composing an Acoustic Model (AM) and a Language Model (LM) offline. The Offline composition simplifies the implementation of a speech recognizer as only one WFST has to be explored for the Viterbi search. However, the size of the composed WFST is huge, typically larger than a Gigabyte, resulting in a large memory footprint and memory bandwidth requirements. On the other hand each of the individual AM and LM WFSTs only require a few tens of MBs.

As the second contribution, we take a completely different approach and develop a hardware accelerator for speech recognition that composes the AM and LM graphs on-the-fly. In this ASR system, the fully-composed WFST is never generated in main memory. On the contrary, only the subset required for decoding each input speech fragment is dynamically generated from the AM and LM models. In addition to the direct benefits of this on-the-fly composition, the resulting approach is more amenable to further reduction in storage requirements through compression techniques. The proposed design, which we call UNFOLD, performs the decoding in real-time using the compressed AM and LM models, and reduces the size of the datasets from more than one Gigabyte to less than 40 Megabytes, which can be very important in small form factor mobile and wearable devices. Besides, UNFOLD improves energy-efficiency by orders of magnitude with respect to CPUs and

GPUs. Compared to the previous accelerator, the UNFOLD ASR system provides 31x reduction in memory footprint and 28% energy savings on average.

In spite of significantly, reducing ASR's memory footprint using UNFOLD, the accelerator still requires 1.5GB/s of memory bandwidth in order to decode the speech signal. However, when using low-power memory technologies this bandwidth requirement is considerably above the peak bandwidth that they provide. Thus, we propose two combined techniques implemented on top of UNFOLD ASR accelerator in order to significantly reduce the energy consumption and memory requirements. First, by leveraging the locality among consecutive segments of the speech signal, we develop a Locality-AWare-Scheme (LAWS) which exploits the on-chip recently-explored data while removing most of the off-chip accesses during the ASR's decoding process. As the second phase, we introduce an approach to improve LAWS's effectiveness by selectively adapting the amount of ASR's workload, based on run-time feedback. In particular, we exploit the fact that the confidence of the ASR system varies along the recognition process. When confidence is high, the ASR system can be more restrictive and reduce the amount of work. The proposed design including both techniques provides a saving of more than 87% in the memory requests and 2.3x reduction in energy consumption, and a speedup of 2.1x with respect to a state-of-the-art baseline design.

As mentioned, a typical ASR system includes a DNN for computing acoustic scores. After optimizing the Viterbi search, DNN becomes the main bottleneck in ASR. Therefore, in order to increase the DNN efficiency, we investigate the effectiveness of pruning schemes, which by removing unimportant connections or neurons, delivers higher performance and energy-efficiency with negligible impact on accuracy. Moreover, we also applied low-precision quantization in order to significantly save the energy-consumption by reducing the number of bits and also simplifying the operations from floating-point to integer. When applitying these well-known optimizations, we observe that the execution time of the ASR system is significantly increased by 33% (for pruning), and 70% (for quantization).

Although pruning or quantization 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. We have seen that, even though top-1 accuracy may be maintained, the likelihood of the class in the top-1 is significantly reduced when using the pruned/quantized models. This lower confidence of the acoustic-scores causes an explosion of the hypotheses in the Viterbi search. Based on this observation, we propose a novel hardware-based ASR system that effectively integrates a DNN accelerator for pruned/quantized models with UNFOLD's 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.

The final solution, with the corresponding hardware support for using the optimized DNN and WFST models, delivers 222x real-time ASR with a small power budget of 1.26 Watt, small memory footprint of 41 MB, and a peak memory bandwidth of 381 MB/s, being amenable for low-power mobile platforms.

Keywords

Automatic Speech recognition, Weighted-Finite-State-Transducer (WFST), Hardware Accelerator, Low-Power Architecture, Real-Time, Large Vocabulary.

Acknowledgement

Firstly, I would like to express my sincere gratitude to my two advisors, Jose-Maria Arnau and Antonio González, for their continuous support of my PhD study and related research, for their patience, motivation, and immense knowledge. I am so much thankful of Prof. Antonio González offering me the opportunity to start a research career at ARCO in UPC and providing me financial support during these years. I am always indebted to Prof. Jose-Maria for all his support and the faith he had in me allowing to develop and build my own infrastructure for my PhD research, and also for giving me the fruitful ideas to help my work gets mature enough. Their guidance helped me in all the time of research and writing of this thesis and the several previous publications. I could not have imagined having better advisors and mentors for my Ph.D study.

Besides my advisor, I would like to thank the rest of my thesis committee: Juan Luis Aragón, Pedro Marcuello, and Jordi Tubella, for their insightful comments and encouragement, but also for their questions which incented me to widen my research from various perspectives.

I owe a very important debt to my dear friend, Hamid Tabani during these years. Hamid has been extraordinarily supportive and optimistic and I am always greatful to him for all his help during the first years of my PhD that I was struggling to adapt to everything from life to work, here in Barcelona. I also want to thank my fellow labmates (the "The D6ers"), Albert and Marc who have helped me through the two papers that I published, Martí-Anglada, Martí-Torrents, Gem, Enrique, Franyell, Dia, Azue, Josue, Oscar, and Dennis, for the stimulating discussions, for the sleepless nights we were working together before deadlines, and for all the fun we have had in the last four years.

I would like to thank my family: my parents and my little sister for supporting me spiritually throughout writing this thesis and my my life in general. I owe my deepest gratitude to my parents and I would like to dedicate this thesis to them. I have been extremely fortunate in my life to have parents who have shown me unconditional love and support. The relationships and bonds that I have with my parents hold an enormous amount of meaning to me. I admire them for all of their accomplishments in life, for their independence and for all of the knowledge and wisdom that they have passed on to me over the years. My parents have played a key role in the development of my identity and shaping the individual that I am today.

Last but not the least, I have to admit that all that kept me running toward the success at my work and also following my PhD research is because of the person who has stood by my side for the past four years and has supported me by her encouragement, positiveness, and never-ending love, my dearest wife, Parinaz. Without doubt, her support, patience and unwavering love were undeniably the bedrock upon which the past four years of my life have been built.

This thesis is dedicated to my parents
for their endless love, support and encouragement.

Contents

List of Figures

List of Tables

1 Introduction

This chapter describes the motivation behind this work, explaining the challenges for deploying ASR systems for mobile and IoT devices and a summary of our contributions in order to remove or ameliorate the system's inefficiencies. Afterwards, we go through some state-of-the-art approaches for ASR design and make a comparison with our proposal to show the key objectives of this thesis.

1.1 Motivation

Nowadays, the interaction with smart devices, such as cellular phones and wearables, has shifted from the traditional keypads and even touch-screens towards more intuitive and yet sophisticated interfaces based on image recognition and speech recognition. Some commercial examples are broadcast news transcription [47, 68], voice search [90], automatic meeting diarization [7] or real-time speech translation in video calls [93]. At the cornerstone of such applications, voice-based user interfaces are becoming increasingly popular, significantly changing the way people interact with computers.

Personal Digital Assistant (PDA) tools like Google Now [2], Apple's Siri [9] or Microsoft's Cortana [65] rely on ASR as their interface. Moreover, a broader adoption of this interaction system is growing with the anticipated popularization of IoT devices and wearables. Thus, companies are actively improving their software in order to allow the users to interface with mobile devices using voice. However, substantial amount of computation power is necessary for enabling speech recognition and natural language processing for large grammars and dictionaries. These requirements make solutions to rely on the Internet to offload most computations to servers in the cloud. For instance, Apple uses a small local neural network to detect "HeySiri" [1], whereas the bulk of Speech-Recognition is done in the cloud. Although this approach is functional, its end-user

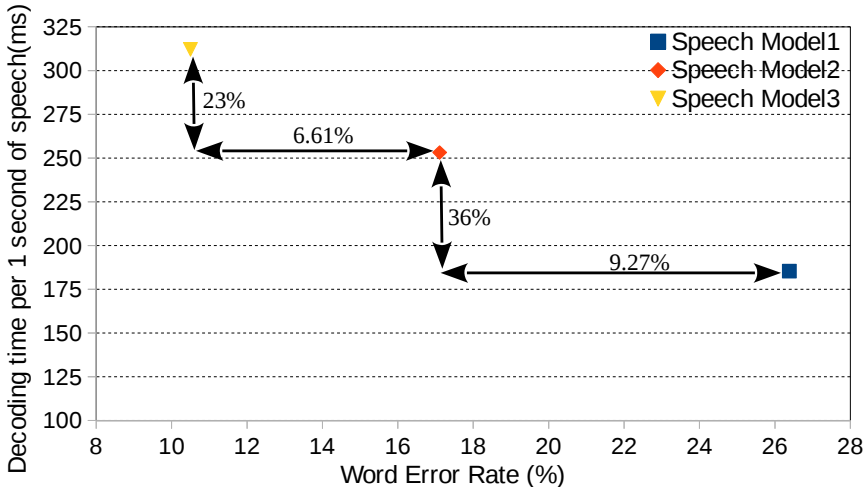


Figure 1.1: Decoding time versus WER for the four different speech models with various complexity. The latency numbers are collected for the GPU (GM204) of Tegra X1 mobile SoC.

experience is negatively impacted by the limited response time in situations of slow Internet connection and the impossibility of interacting with the device if the user is in a place with no Internet coverage.

Voice-based interactive applications must provide fast and highly-accurate response so as to meet the user's satisfaction. It must deliver large-vocabulary, real-time, speaker- independent, continuous speech recognition [52, 110]. Unfortunately, supporting fast and accurate speech recognition comes at a high energy cost, which in turn results in fairly short operating time per battery charge. By performing ASR remotely in the cloud, we may potentially alleviate this issue, but as aforementioned, it comes with its own drawbacks while increasing the latency due to the time required to transfer the speech and the energy consumption of the communication subsystem.

We have evaluated ASR performance on both mobile CPU and GPU platforms. However, we choose the GPU as our baseline since it delivers significantly faster execution of around an order of magnitude compared to the CPU. Figure 1.1 shows the Word-Error-Rate with respect to the amount of computation time in a mobile GPU (GM204). The time is displayed as the Real-Time-Factor (RTF), which is the ASR decoding time for 1 second of speech (in milliseconds). As the figure shows, the higher the accuracy of the ASR system, the higher the time required to decode the speech. Regarding the most accurate model, the GPU has 312 ms latency, which can get much higher as the complexity of newer models grow [106, 105, 3]. Moreover, the GPU consumes a considerably large power budget of several Watts to achieve real-time decoding. On the other hand, the real-time performance can get even more challenging for wearable and IoT devices as they are battery-operated. Given the issues with the software-based solutions running locally or in the cloud, we believe that hardware acceleration represents a better approach to achieve high-performance and energy-efficient speech recognition in mobile devices.

In order to achieve a low RTF for the ASR system, we compare the different solutions running the most accurate model of Figure 1.1. Figure 1.2 shows the power dissipation versus the RTF for the different platforms, desktop and mobile GPUs and our most efficient acceleration design. As we can see, the desktop GPU performs much faster than the mobile one, whereas consuming

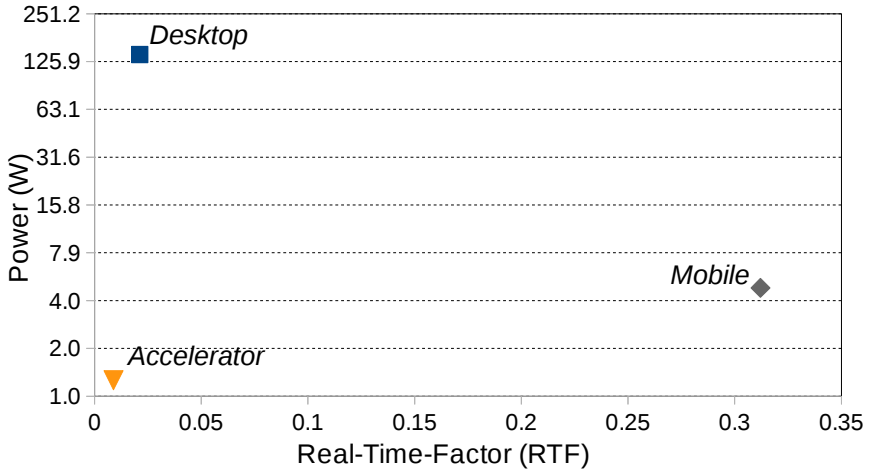


Figure 1.2: The power dissipation versus Real-Time-Factor (RTF) for running Kaldi ASR on Geforce GTX980 Desktop GPU, Tegra-X1 mobile GPU, and our accelerator.

an order of magnitude (30x) higher power. On the other hand, the accelerator delivers the most efficient tradeoff by significantly outperforming both GPUs and requiring nearly 4x less power than the mobile GPU.

In this thesis, we focus on improving performance and energy-efficiency of ASR for mobile devices. ASR is already an essential feature for smartphones and tablets. Furthermore, speech technology will be of special interest for wearable computing, becoming a hard requirement for most mobile devices such as smart watches. Moreover, the future speech models are likely to get more complex [13, 14, 6] in order to provide better accuracy, which makes it even more challenging to achieve real-time performance. Due to their tiny form factor, wearable devices are extremely constrained in terms of area and power and, therefore, they cannot afford complex hardware solutions like GPUs. Hence, we believe there is a strong case for real-time and energy-efficient ASR on mobile platforms, which is the focus of this work.

1.2 Problem Statement, Objectives and Contributions

The objective of speech recognition is the transcription of acoustic signals into a sequence of words. A state-of-the-art ASR pipeline consists of three main stages which are *Feature Extraction*, *Acoustic Scoring* and *Search Engine*. Figure 1.3 shows an overview of the ASR pipeline. First, the input audio signal is split in frames, where each frame represents a 10 ms interval of the speech signal. Next, the *Feature Extraction* stage transforms the digitized audio samples within a frame into a vector of features. These features are then related to phonemes by an *Acoustic Model* in the *Acoustic Scoring* stage. The baseline ASR system employed in this thesis uses different schemes such as Gaussian Mixture Models (GMMs) [83], Deep-Neural-Networks (DNNs) [115], and Recurrent Neural Networks (RNNs) [63] for acoustic scoring. Afterwards, the *Search Engine* finds the most likely sequences of words, called word-lattice, by executing a Viterbi beam search on the combined language+accoustic model. Finally, by using a backtracking step, the utterance is extracted from the lattice generated by Viterbi search. The last two steps, i.e. the acoustic-scoring

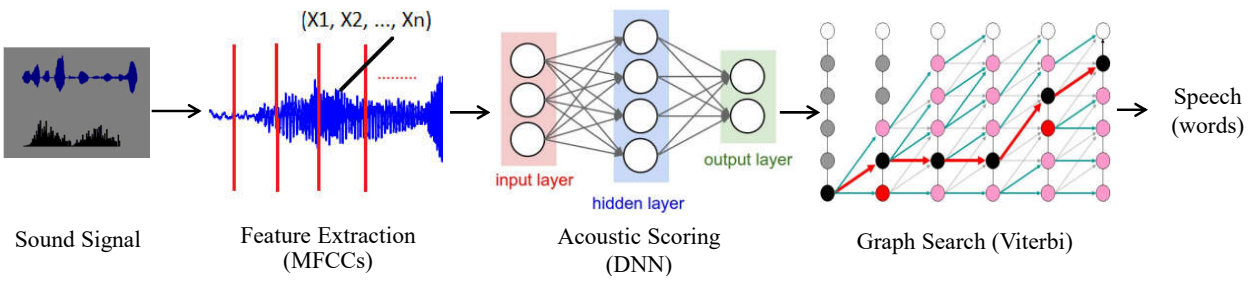


Figure 1.3: An overview of the ASR pipeline. After receiving the sound signal, it is partitioned into fragments. Then, they are transferred into MFCC feature-vectors, which are fed to an acoustic-scoring system. In here, we use a DNN as the acoustic-scoring phase. Finally, the likelihoods of speech phonemes are used in a graph search, called Viterbi, to generated the most probable sequence of words.

and the Viterbi search, are the main time- and energy-consuming parts (>90%) of an ASR system. Thus, in this thesis, we focus on optimizing them in order to deliver a highly efficient execution of speech recognition. We devote Chapter 2 for presenting a detailed background on the algorithms used for automatic speech recognition.

In this thesis, we pursue one main objective: optimizing the ASR system for small-form battery-operated devices in terms of performance and energy-consumption, while maintaining high accuracy. To this end, we have evaluated a variety of ASR systems with different speech models' complexities. Most of the early previous works have concentrated on simple tasks, such as resource management [82] with a small vocabulary (1500 words) and a simple grammar. These systems are as simple as detecting some codes, i.e. the commands in the form of words or phrases, which correspond to the submission of different tasks. On the other hand, later works are focused on more difficult applications such as decoding life speech from reading the Wall Street Journal task [73], or the broadcast news [35]. These systems have been trained to recognize long sentences of a larger vocabulary of 5K-20k words, spoken by different speakers. Lately, one of the most challenging problems is the decoding of conversational speech transcripts considering very-large vocabulary of 120K-200K words, which is specially difficult due to the nature of the speech considering noise, informality, hesitations, and self-corrections [34, 24].

Recently, ASR advancement has reached human parity for conversational speech recognition by providing nearly 5% of error as reported by Microsoft ASR systems [106, 105]. In order to achieve this accuracy level, these systems use complex speech recognition models with very a large vocabulary of hundreds of thousands of words. In general, ASR systems have a memory footprint in the order of gigabytes and a peak memory bandwidth usage of 10 Gbytes/s [109]. Such a high degree of accuracy using these complex systems comes at the expense of large storage requirements and computational cost.

Despite the progress in ASR accuracy, there are still a lot of challenges remaining in order to provide an efficient solution for low power mobile and wearable devices. As depicted by Figure 1.3, ASR includes a heterogeneous pipeline whose stages have different resource requirements due to their particular characteristics. DNN, for the acoustic scoring, shows a completely predictable computation pattern which is independent of the input signal's features. In addition, it involves a lot of parallel computations (multiply-and-add) for evaluating the audio phonemes' probability spectrum, and hence, it generates a lot of memory requests in order to fetch the millions of DNN's

weight parameters. On the other hand, Viterbi search is a more irregular algorithm that depending on the input probabilities produces a different expansion of the search graph. Therefore, the ASR system requires a careful design for the different parts of its pipeline, as well as an effective interaction among them, in order to obtain an efficient high-performance evaluation with low energy-consumption.

As discussed earlier, CPU and GPU based approaches deliver inefficient solutions for ASR. Figure 1.1 shows the main problem with the mobile GPUs, which may deliver a real-time response for the simple speech models, however, they show a significantly lower performance as the complexity of the model increases. Furthermore, by the emergence of the new technologies which require a highly-accurate ASR as their critical component, such as Skype online translator [94] and Machine Comprehension [91], the need for a more efficient alternative becomes crucial as they necessitate a performance several times faster than real-time. Even though mobile GPUs provide a highly parallel architecture that can perform ASR an order of magnitude faster than CPUs, as illustrated in Figure 1.4, it does not provide the most efficient solution for the whole ASR, failing to exploit all its parallelism for the Viterbi search. Moreover, in order to achieve such performance benefits, a mobile GPU requires a huge power-budget, in the order of hundreds of Watts (30x higher than a mobile CPU).

Given all the challenges mentioned above, and also the inefficiency of the previous software approaches, we focus on customized architectures using hardware acceleration. In this thesis, we attack all the aforementioned problems in the ASR system, by optimizing the main ASR's pipeline stages, i.e. Viterbi seach and DNN evaluation. Furthermore, we propose a new representation of the ASR dataset that reduces its memory footprint memory bandwidth requirements by a large extent.

As the first step, after characterizing the main bottlenecks of the system, we design an accelerator for Vitebi search which takes almost 90% of the decoding time in the mobile GPU. Using the accelerated design, we reduce the search time by 1.7x and 22x with respect to a high-end GPU and a mobile GPU, respectively, while saving energy by a large margin (287x and 48x).

Since ASR requires a huge graph to represent the combined acoustic and language model, our second target is to decrease the memory requirements of this system. To do so, we propose a completely different scheme of using the two unfolded graph models, rather than a vast unified, fully-composed graph, reducing the memory footprint by 10x. In addition, we introduce several compression techniques for each graph that all together result in around 31x compression ratio. Furthermore, we implement a locality-aware scheme on top of our final design, in order to lessen the memory bandwidth of the accelerator by 3.8x.

After optimizing the Viterbi search, we concentrate on the other part of ASR which is the DNN acoustic-scoring, as it now becomes the main bottleneck of the system. We include some of the state-of-the-art acceleration architectures. Theses accelerators employ very low numerical precision using linear quantization and DNN pruning to increase the performance and reduce the energy-consumption. Even though it has been reported that these optimization techniques provide large benefits without losing any accuracy for isolated DNNs, we make the observation that they have a negative side-effect for the applications that use the DNN scores for a second phase, such as for Viterbi search in ASR. We have observed that they cause an exponential grow in the Viterbi's

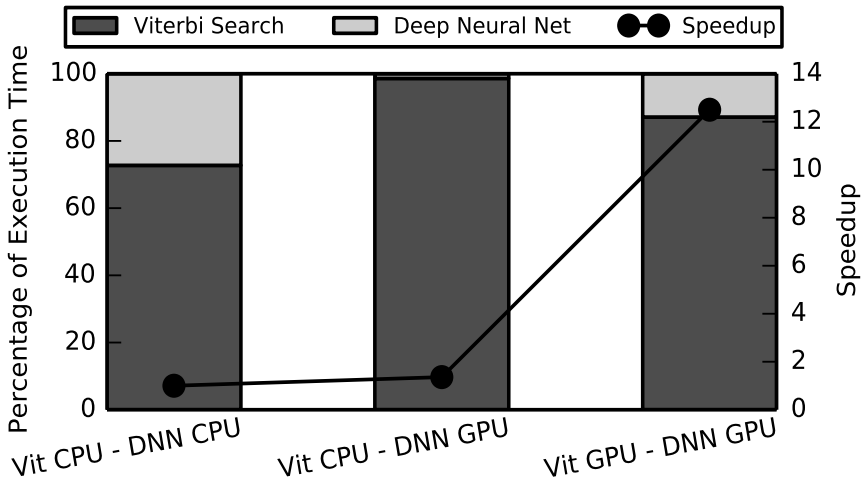


Figure 1.4: Percentage of execution time for the two components of a speech recognition system: the Viterbi search and the Deep Neural Net. The Viterbi search is the dominant component, as it requires 73% and 86% of the execution time when running on a CPU and a GPU respectively.

workload. As the DNN's output becomes less confident, the Viterbi search has to expand many more alternatives to decode the speech. In order to solve this issue, we propose to modify some part of the Viterbi accelerator's microarchitecture, in order to restrict the search expansion to the N-best decoding paths for each speech frame. In the end, the fully-accelerated ASR system achieves up to 4.2x speedup and 9x energy savings with respect to the previous solutions.

The following sections outline the problems we are trying to solve, describe the approach we take to solve each problem and highlight the novel contributions of this thesis.

1.2.1 Viterbi Acceleration

Figure 1.4 shows the breakdown of ASR's decoding-time for several hours of speech from the English language dataset of Kaldi [76], a well-known ASR toolkit, on both a high-end desktop CPU and a GPU. Viterbi search represents the main performance bottleneck requiring more than 73% of execution time in all scenarios. Furthermore, this share becomes more crucial in cases that GPU runs DNN (86%-99%) due to exploiting high parallelism for the neural-network's execution. Since the Viterbi algorithm is hard to parallelize [22, 51, 61], it is the dominant component when running the whole ASR on GPUa. Therefore, a software implementation cannot exploit all the parallel computing units of modern multi-core CPUs and many-core GPUs.

Based on these observation, we recognize Viterbi search as the main time and energy consuming part of the ASR system. So, we tackle this problem by offloading the most time-consuming part of an ASR system to an Applcation-Specific Integrated Circuit (ASIC) accelerator. To do so, we design an customized architecture, which employs a specialized pipeline for performing the search expansion. Moreover, we present several techniques in order to improve the accelerator's memory subsystem: first, a prefetching scheme tailored to the characteristics of the Viterbi algorithm, and second, a memory-bandwidth saving technique that removes 20% of the off-chip accesses. Figure 1.5 summarizes the main benefits of the proposed design in terms of performance and energy consump-

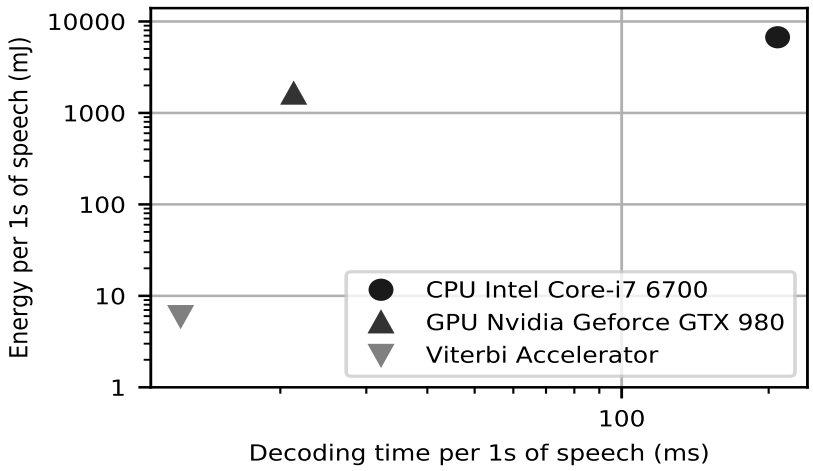


Figure 1.5: Energy vs decoding time per one second of speech, for Viterbi accelerator, CPU, and GPU.

tion. As depicted, Viterbi accelerator outperforms CPU by orders of magnitude and achieves an speedup of 1.7x compared to the GPU, while reducing energy consumption by two orders of magnitude (287x). Chapter 4 extensively describes the accelerator's design, architectural analysis, and the experimental results. This work has been published in the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2016 [110].

As a continuation of the work on Viterbi acceleration, we integrate the whole ASR application's pipeline by using the accelerator combined with a mobile GPU (GM204) embedded as a part of NVIDIA Tegra-x1 SoC. The GPU and the accelerator work in parallel in a pipelined manner. While the GPU computes the DNN for the next frame of the speech signal, the accelerator performs the Viterbi search for the current frame. Therefore, we overlap the latency of the DNN and the Viterbi search. Compared to a GPU-only system, our hybrid scheme improves performance by 5.25 times while reducing energy by 2.05 times. This work has been published in the IEEE Micro Journal [111], 2017.

1.2.2 Memory-Efficient ASR System

Another challenging issue of ASR is the huge memory requirements due to the vast size of speech dataset. Figure 1.6 shows the memory footprint breakdown for different ASR decoders using either GMM, DNN or RNN. As illustrated, the graph model, called as Weighted Finite State Transducer (WFST), is clearly the largest component, requiring between 87% and 97% of the total memory for ASR, which represents more than one Gigabyte for large vocabulary systems.

WFST compression [81] can ameliorate the problem of huge memory footprint, but still several hundreds of Megabytes are required for the WFST. In order to deal with the excessive memory requirement of large vocabulary ASR, we analyze how the WFST is constructed during training. The WFST is created from two knowledge sources: Acoustic Model (AM) and Language Model (LM). Both AM and LM are represented as WFSTs, and then WFST composition is used to generate a unified WFST [66]. Offline composition simplifies the decoding stage, as only one WFST has to be searched. However, it requires a multiplicative combination of states and arcs

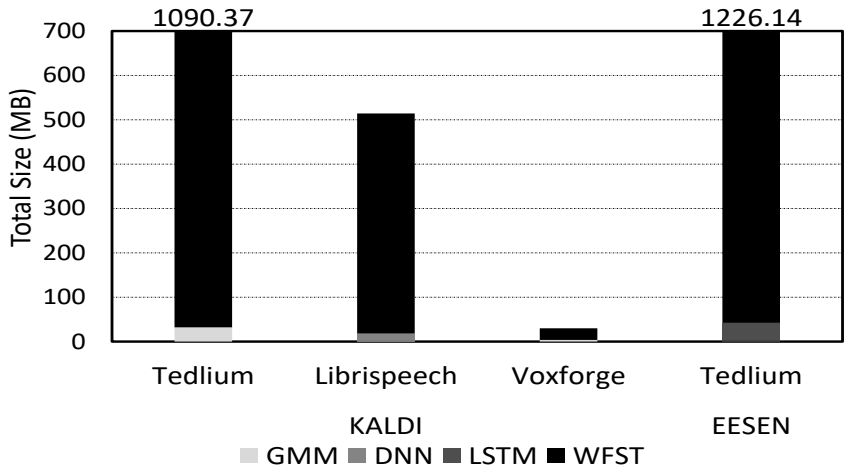


Figure 1.6: Sizes of the different datasets employed for ASR. The WFST is by far the largest component.

compared to the AM and LM graphs, resulting in the large sizes reported in Figure 1.6.

In this work, we propose a Viterbi accelerator, called *UNFOLD*, that dynamically composes the AM and LM graphs during the decoding, avoiding the explosion in WFST size produced by offline composition. On-the-fly composition [18, 17, 46] reduces memory storage requirements by a large extent since the total size of the AM and LM graphs is significantly smaller than the size of the composed WFST. To further reduce memory footprint, we show that these individual WFSTs can be compressed very efficiently, in a more effective manner than the composed WFST. The combined use of on-the-fly composition and compression reduces the size of the datasets by 31x on average for several ASR systems. As a result, *UNFOLD* performs the decoding by using less than 40 Mbytes of storage instead of more than one Gigabyte.

The only drawback of on-the-fly composition is the increase in activity during the decoding, since WFST composition is moved from the training to the decoding stage. To achieve real-time performance, *UNFOLD* includes simple yet efficient hardware support for composition operations. On the other hand, the large reduction in memory footprint results in smaller miss ratios in the caches of the accelerator, which significantly reduces the memory bandwidth usage, and is highly beneficial for performance and energy. In other words, on-the-fly composition increases the amount of computations and on-chip memory accesses, but it drastically reduces the number of expensive off-chip main memory accesses. Since external DRAM accesses require orders of magnitude more energy than local on-chip operations [49, 42], *UNFOLD* achieves not only a large reduction in memory footprint, but also 28% energy savings on average with respect to the initial design [110], while achieving real-time performance by a large margin (155x). Chapter 5 explains *UNFOLD's* architecture and provides the details of our analysis on various benchmarks. This work has been published in the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2017 [108].

Despite of the benefits gained by UNFOLD, there is still one vital requirement which prevents the accelerator from being entirely applicable for the small form-factor wearable and IoT devices: the memory bandwidth of almost 16 Gb/s. These devices normally use low-power memory technologies with limited throughput, such as NAND/NOR flash memories [26]. As reported by

Micron, these memory systems can achieve a maximum bandwidth range between 1 to 6 Gb/s [64]. Consequently, we need to significantly improve the ASR's memory management for these devices. Furthermore, as each DRAM memory access consumes nearly three orders of magnitude more energy than a typical computation or on-chip memory accesses [39], high memory requirements are also the main energy bottleneck in ASR systems.

In order to overcome this challenge of the UNFOLD's full applicability, we introduce a Locality-AWare Scheme (LAWS). LAWS manages to control the Viterbi search space by combining the likelihood score and the data locality of the explored search paths, i.e. the search hypotheses. In other words, LAWS uses a dynamic beam adaptation policy for limiting the number of hypotheses to the ones with higher probability than the beam. This policy not only takes into account the locality of the data, besides the likelihood scores, but also the confidence of the ASR system at each frame. LAWS achieves more than 2x improvement in both performance and energy consumption, while maintaining accuracy and reducing the main memory activity by more than 7.9x. Chapter 5 elaborates on the LAWS mechanism with the modification made to the *UNFOLD's* architecture.

1.2.3 Combining DNN and Viterbi acclerators for ASR System

After optimizing the Viterbi search component of the ASR system, the DNN acoustic-scoring becomes the main ASR bottleneck. Thus, we have leveraged some of the previously proposed accelerator's designs [87, 39] in order to speed up the whole ASR's pipeline. Furthermore, we have applied low-precision quantization [8, 53, 54, 28] and DNN pruning [42, 41, 114, 39] techniques, two effective solutions to further increase the performance and save the energy-consumption of the DNN. Even though DNN pruning and quantization have a negligible impact on the ASR accuracy, we encounter an important side-effect due to the confidence reduction in the DNN predictions. We measure the confidence as the probability assigned to the class with maximum likelihood, i.e. the top-1 class. The confidence loss results in a significant increase of the ASR workload which cause a considerable slowdown in the performance.

Figure 1.7 shows the distribution of the DNN scores for the original and three pruned models of a state-of-the-art DNN for speech recognition [116], for a particular frame. The pruned models have 70%, 80% and 90% of pruning ratios, using a state-of-the-art pruning scheme proposed by Han et al. [42]. This DNN generates the likelihoods for 3482 classes, that correspond to different sub-phonemes of the language. 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. Furthermore, we have observed a similar trend for the quantized DNN models.

Even though the Word Error Rate (WER) is maintained using the aforementioned DNN optimization techniques, Viterbi search, as the consumer of the DNN's output's probability, suffers from a significant increase in the number explored hypotheses. Viterbi search explores multiple

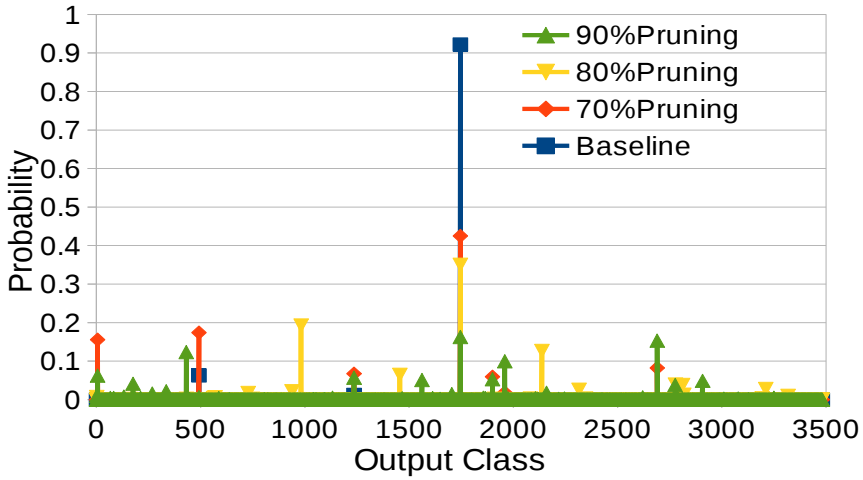


Figure 1.7: Distribution of scores for a DNN for speech recognition and three pruned version at 70%, 80% and 90% of pruning. Although pruned models correctly identify top-1 class, the distribution of likelihoods is severely affected and confidence is largely reduced.

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 most-likely sub-phonemes (classes in top-1) get very high scores whereas the hypotheses using least-likely 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 1.7), 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 1.7), more paths are explored.

According to our experiments, in spite of minimizing the DNN computation by using DNN pruning or low-precision quantization, the ASR system preserves the accuracy; however, it causes a significant slowdown in the Viterbi search. Figure 1.8 shows the execution time and accuracy of our ASR system that employs a DNN accelerator [87] and *UNFLOD* [108], as the Viterbi 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 accelerator, it introduces a large slowdown in the Viterbi search. The increase in Viterbi search execution time offsets part of (or all) the benefits of the DNN pruning, and it results in 33% slowdown for 90% of pruning.

In order to tackle this issue, the straight-forward 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, and

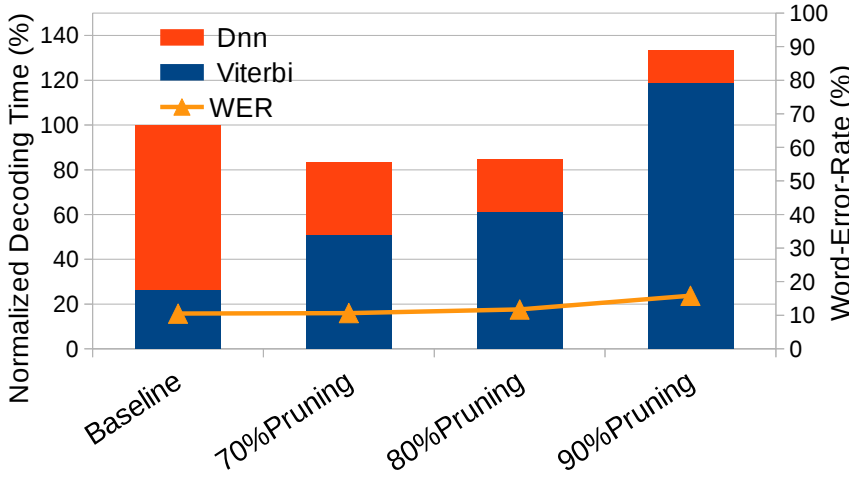


Figure 1.8: Normalized execution time and Word Error Rate (WER) for a stateof-the-art ASR system, using a DNN with different degrees of pruning: 0(Baseline), 70%, 80% and 90%. Pruning has a large impact on the execution time of the Viterbi beam search.

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. Chapter 6 presents in more detail the implementation of our scheme considering the modifications made to *UNFOLD's* microarchitecture. This work has been published in the 45th Annual IEEE/ACM International Symposium on Computer Architecture (ISCA), 2018 [107].

1.3 Related Work on Accelerated ASR System

There are two main approaches for the ASR system's implementation. The first one, called hybrid scheme, combines a DNN for acoustic scoring with the Viterbi beam search, which is the state-of-the-art scheme in speech recognition [110, 108, 79]. Another emerging approach for ASR is the so-called end-to-end solution with a Recurrent Neural Network (RNN) [36, 37], typically using a Long Short Term Memory (LSTM) network [45]. This approach merges an RNN with a beam search on a Language Model WFST to achieve an accuracy comparable to the hybrid systems [63, 37]. We first describe several systems using the hybrid ASR mechanism and second, we go through some end-to-end systems.

1.3.1 Hybrid ASR Systems

There have been several attempts at optimizing and accelerating ASR application using various computing platforms, including GPUs, FPGAs or ASICs.

GPU-Based Implementations

Regarding the Viterbi search, Chong et al. [113, 22] proposed an implementation in CUDA that achieves 3.74x speedup with respect to a software decoder running on the CPU. In this work, we consider the CUDA implementation as our baseline and show that our accelerator, which is specifically designed for speech recognition achieves an energy reduction of two orders of magnitude with respect to the GPU. The GPU is designed to perform many floating point operations in parallel and, hence, it is not well-suited for performing a memory intensive WFST search.

Nonetheless, we have observed good performance speedup for the DNN part of ASR on a GPU (26x), as it is more computationally intensive and GPU excels at that [4]. Therefore, in this work we have investigated a GPU-accelerated system, in which a GPU performs the DNN acoustic-scoring and an accelerator operates on the Viterbi search. In this implementation, the GPU and the accelerator work in parallel in a pipelined manner. While the accelerator executes the Viterbi search for the current frame of speech, the GPU computes the DNN for the next one. Thus, we overlap the latency of the DNN and the Viterbi search.

FPGA and ASIC Implementations

By using a FPGA or ASIC-based accelerator, we can customize its design according to the ASR's requirements. However, since ASR consists of two phases with different characteristics, it requires different acceleration components dedicated to each part. We divide the related work on ASR's acceleration based on these two phases.

Viterbi Search

Choi et al. [112, 21] present an FPGA implementation that can search a 5K-word WFST 5.3 times faster than real-time, whereas Lin et al. [58] propose a multi-FPGA architecture capable of decoding a WFST of 5K words 10 times faster than real-time. Their use of a small vocabulary of just 5K words allows them to avoid the memory bottlenecks that we have observed when searching large WFSTs. Considering the very large-vocabulary ASR systems, we focus on designing the accelerator's implementation in a way to handle a more efficient memory management in order to better exploit all the hardware resources and remove the long latencies imposed due to the memory fetches. Thus, the work of this thesis is different in the way that we deal with significantly more complex systems that demand much higher memory requirements than the small vocabulary systems.

ASIC approaches for accelerating speech recognition in hardware have attracted the attention of the architectural community in recent years. Price et al. [80] have presented several versions of ASR acceleration designs based on different acoustic-scoring models, such as GMM [78] and DNN [79, 81]. These hardware solutions integrate the whole ASR pipeline including feature-extraction, acoustic-scoring, and Viterbi search. However, in our approach we decide to use a different hardware architecture for each pipeline's stage, since they have particular characteristics considering their computational cost and memory and power requirements. Moreover, this allows us to reuse the

different parts of our implementation in the newer model with an increased level of complexity. For instance, in a recent approach for speech recognition [75], the ASR system obtains human-level accuracy by merging a DNN for the acoustic-scoring with a Viterbi beam search that produces a lattice of the most probable hypotheses, which are then rescored with an RNN model. In the following, we go over the architecture presented in [80], with an extensive comparison with our proposal.

Regarding the feature-extraction, we have found that it takes less than 1% of the whole ASR's execution time, and it does not pay off to use an specialized hardware to perform that. Therefore, it can be executed in a conventional CPU.

The acoustic-scoring is the second most time-consuming step of the ASR, taking 26% and 15% of the total execution on CPU and GPU, respectively. In [79], a DNN accelerator is designed based on a simple fixed-function SIMD architecture that computes DNN inference of multiple frames in parallel. Regarding the memory interaction, it uses a compressed model to reduce the bandwidth requirements. However, it does not support neither DNN pruning nor low-precision quantization. On the other hand, we leverage state-of-the-art solutions in DNN acceleration considering the issues raised by the load-imbalance and memory alignment caused by DNN-pruning and quantization, respectively. Moreover, we investigate the interaction of DNN and Viterbi accelerators, identify the negative effects that cause an explosion in the ASR workload, and design a technique avoid their penalty.

The Viterbi accelerator's implementation in [80] is different from our proposal, especially in the memory architecture. For instance, we include several caches to speedup the memory accesses for different data types, whereas they only use one cache for the entire speech graph dataset. Since the memory subsystem is the main bottleneck for the ASR systems, we propose a new prefetching scheme that accurately predicts the access pattern based on a decoupled access-execute architecture. Furthermore, we introduce a memory bandwidth saving technique that reduces the memory fetches by 20%. The reason of the simplicity in their design is that it targets to perform close to real-time as it necessitates very low power consumption, between 172 uW to 7.78 mW.

ASR is normally part of a larger application which uses its result for other purposes such as Skype online translator [94], speech synthesis [102], and machine translation [11]. Consequently, the acceleration design should be at least several times faster than real-time in order for the whole application's processing to finish in real-time. Therefore, our proposed design provides a much faster solution while performing more complex ASR systems that deals with larger vocabulary (200K words). Additionally, we propose a low-power technique based on a drowsy cache structure which significantly reduces the leakage power of the accelerator's on-chip buffers, which are the main power consuming blocks of our system.

Price et al. [80] have also focused on reducing the main memory pressure for ASR systems, by aggressively beam pruning the Viterbi search in order to decrease the search space. Price et al. [80] have introduced the "hard" absolute pruning scheme, which tries to restrict the number of hypotheses to less than 8K. When increasing the number of hypotheses above the limit, the "hard" cutoff is updated, possibly several times, followed by a repruning step to decrease the number of hypotheses. However, the repruning can happen several times and therefore becomes time-consuming. In our proposal, we introduce a locality-aware scheme for Viterbi search that can

reduce the search space by 60%. Furthermore, we define a new metric as the search confidence based on the number of hypotheses, so as to dynamically select the beam width. In the end, we decrease the memory activity by 7.9x with negligible loss in ASR accuracy.

Regarding the reduction of the speech dataset, there have been several simple straight-forward compression techniques evaluated in [81]. That work tries to reduce the number of bits for representing the information of the arcs and states of the speech graph. They report between 0.279 and 0.366 compression ratio for the different ASR decoders. In comparison, we take a different approach by decomposing the vast speech graph into two basic graph-based knowledge sources, which reduces the size by 10x. Moreover, we take advantage of some simple compression techniques for each individual graph based on their particular properties, which finally results in 0.032 compression ratio, achieving 8.8x improvement over the previously proposed compression techniques.

There have been several other proposals which target high-rate and low-power ASIC designs. Johnston et al. [52] present an accelerator for 60K-word vocabulary WFST-based ASR model that performs at 127x real-time, and dissipates around 454 mW power. In comparison to this work, we have extensively studied the main bottlenecks with the memory subsystem which have led us to a better implementation of the cache architecture. Moreover, we work on a different approach as we perform the Viterbi search using two model grapsh rather than one, which makes the search expansion more complicated and computationally expensive. Finally, we achieve up to 155x real-time performance speedup for a system with 3.3 times larger vocabulary, requiring 512 mW power budget.

DNN Acoustic Scoring

There have been several proposals regarding the DNN accelerations, which can also be applied to the ASR's acoustic-scoring part. These architectures can be divided into two main groups: sparse and dense accelerators. Regarding the dense models, Riera et al. [87] present an architecture based on DianNao [20], which tries to avoid unnecessary computations by exploiting similarity between different inputs. It evaluates the speech network at nearly an order of magnitude faster than Nvidia GTX 1080 GPU. Since DNNs are both computation and memory intensive, other architectures, such as [86, 53, 54] try to optimize the DNN evaluation by reducing the data precision, which results in very low-power and energy-efficient execution.

A common technique to reduce DNN power is the pruning [42, 41, 40] 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. EIE [39] and Scalpel [114] are two examples of DNN sparse accelerators which use a customized architecture based on an efficient sparse data representation, in order to give the performance and energy efficiency promised by the removed computations. In this work, we analyze the interaction between DNN and Viterbi accelerators, considering all the optimization techniques. In the end, we propose a novel solution that while keeping all the previous improvements, it mitigates all the drawbacks of the integration, operates two orders of magnitude (222x) faster than real-time [107], and requires a rather small memory (41.4 MB) for both the DNN and Viterbi search.

1.3.2 End-to-End ASR Approach

End-to-end RNN-Based approaches have recently improved significantly. However, as claimed by the latest Kaldi's results [3], the latest DNN-based hybrid approaches still achieve the best accuracy (3%-4%). On the other hand, in order to achieve the best accuracy for end-to-end systems, they require a beam search to correct the many misspelled words in the decoded transcription. For example, Baidu's Deep- Speech [44] combines an RNN with the Viterbi search algorithm as described in [43], whereas EESEN [63] employs a Viterbi search driven by the scores computed by an LSTM network [40], [92]. Therefore, as the main focus of this thesis on accelerating this part of ASR's pipeline, it can also be applied for these systems.

Regarding the end-to-end ASR systems, there have been several implementations on FPGA [40, 19, 57] and ASIC [92]. Chang et al. [19] present a low-power FPGA accelerator targeting the mobile segment. It implements a small LSTM network and dissipates 1.9 W. Another low-power design has been presented in [57], which consumes 9 W and supports larger models by using aggressive weight quantization. However, due to the large size of the complex, accurate models and the non-negligible increase in Word Error Rate by quantization, they cannot be stored in local memory. ESE [40] is also an FPGA-based ASR sparse accelerator that operates on a 12-bit quantized, pruned LSTM model. It achieves 120x real-time performance while dissipating 41 Watt power. On the other hand, E-PUR [92] presents an ASIC-based accelerator which supports large LSTM networks of hundreds of Megabytes, while using small on-chip storage and low memory bandwidth. It performs 18.7x faster than a mobile GPU, while consuming less than 1 Watt power.

1.4 Thesis Organization

The remainder of this thesis is structured as follows:

Chapter 2 provides basic background information on the speech recognition algorithms. We mainly describe the ASR toolkit that we employ in this thesis and we detail its different stages. Moreover, we describe the various types of model graphs, a.k.a WFSTs, used in this work.

Chapter 3 goes through our experimental methodology. We describe the different tools used in order to evaluate the performance, power, area, and energy-consumption of the system. Moreover, we define the various speech datasets that we use for the ASR accelerator's evaluation.

In Chapter 4, we first evaluate the ASR pipeline on CPUs and GPUs in order to identify the main bottleneck of the different decoders. Second, we present an ASIC-based accelerator's architecture for the Viterbi search part, which takes the majority of execution time. Third, we introduce some techniques in order to optimize the memory subsystem of the accelerator. Finally, we present a thorough experimental evaluation to show the main improvement of our proposed ASR system.

In Chapter 5, we further optimize the Viterbi accelerator to significantly reduce the memory requirements by using on-the-fly WFST composition and compression techniques. Moreover, in order to improve the performance, we include some simple yet efficient hardware supports for

composition operations. Furthermore, we propose a locality-aware scheme, that can decrease the memory bandwidth by 3.8x by removing 87% of the memory requests.

Chapter 6 presents a completely accelerated ASR system that consists of a DNN accelerator for the acoustic-scoring and *UNFOLD* as the Viterbi search accelerator. By analyzing the interaction between the two sub-systems, we define several important characteristics which significantly impact the ASR's performance and energy-consumption. Finally, we propose a modified version of *UNFOLD* that can solve all these interaction issues.

Finally, Chapter 7 outlines some of the future steps and open research areas and summarizes the main conclusions of the thesis.

2

Background on Speech Recognition

In this chapter, we provide background on the Automatic Speech Recognition (ARS) pipeline and algorithms. As approaches based on Weighted Finite State Transducers (WFSTs) have been proven to provide significant benefits over Hidden Markov Models (HMMs) [113, 55], especially for hardware implementations [112], we focus on these ASR systems. We firstly describe an overview for the ASR mechanism. Next, we elaborate on each phase of the decoding process including: feature-extraction, acoustic-scoring and search-engine based on the WFST graph.

Speech recognition is the process of identifying a sequence of words from speech waveforms. An ASR system must be able to recognize words from a large vocabulary with unknown boundary segmentation between consecutive words. The typical pipeline of an ASR system works as follows. First, the input audio signal is segmented in frames of, typically, 10 ms of speech. Second, the audio samples within a frame are converted into a vector of features using signal-processing techniques such as Mel Frequency Cepstral Coefficients (MFCC) [103]. Third, the acoustic model, implemented by either a Deep Neural Network (DNN), Gaussian Mixture Models (GMMs) or a Recurrent Neural Network (RNN) transforms the MFCC features into phonemes' probabilities. Finally, the Viterbi beam search explores on one or multiple WFSTs in order to find the most likely sequence of words [84] from the sequence of phonemes' probabilities.

The input audio signal that represents the speech is segmented into several frames and, next, a vector of features is computed for each frame. These features are the main components that represent the input signal. A DNN consisting of several hidden layers is then applied to detect the most likely sound phoneme from each frame of the speech waveform. Therefore, we will have a spectrum of the phonemes' probability which is inserted into a graph search that decodes the utterance. The graph search, called as Viterbi algorithm, explores the WFSTs, i.e. graph-based speech dataset, in order to find the most likely path or hypothesis that best resembles the audio frames and their corresponding phonemes' likelihood. As illustrated in Figure 1.3, during the search

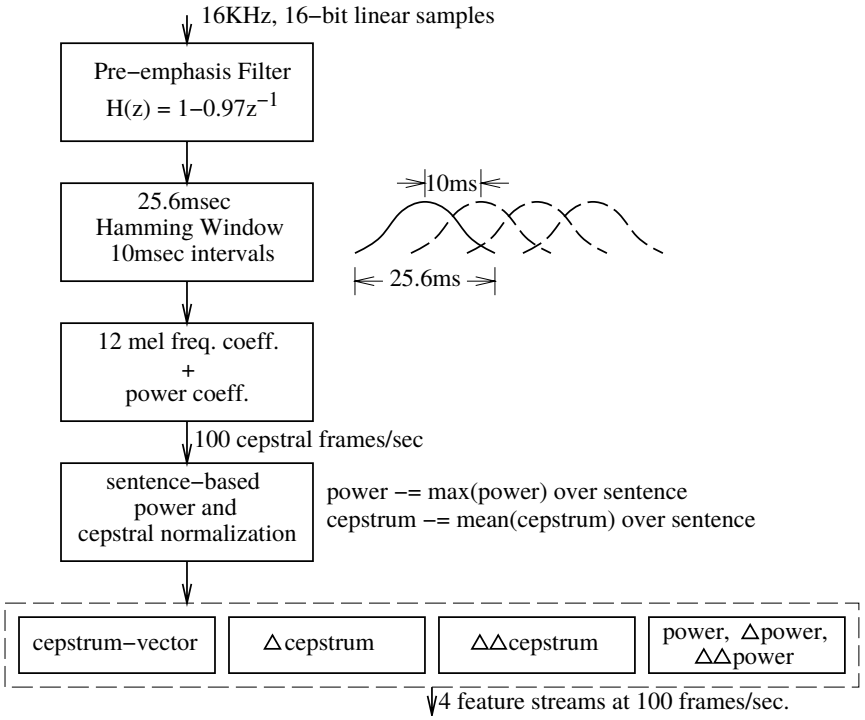


Figure 2.1: An overview of signal-processing for feature-extraction [85].

process Viterbi generates a dynamically-explored graph from the WFST graphs, which is finally backtracked to locate the best path, i.e. the most likely sequence of words according to the speech models. In the following sections we elaborate on each part of the ASR pipeline.

2.1 Feature-Extraction Using Signal Processing

The first step of an ASR system is the computation of feature vectors from the raw speech signal. A speech signal has relatively redundant information as there is high correlation between its adjacent segments [97]. Thus, ASR systems mostly use a parametric representation rather than the whole speech waveform itself. Doing so, we can extract the waveform's data compactly, and simplify the computation for both training and decoding. MFCC, introduced in [15], is one of the most famous sets of parameters used in speech recognition systems.

Figure 2.1 shows the block diagram of the overall processing for computing the signal coefficients. In summary, the vector of 16-bit speech samples (at 16KHz rate) is converted into 12-element Mel-scale frequency cepstrum vectors and a power coefficient in each 10 millisecond frame. We represent the cepstrum vector at time t by $X(t)$ (individual elements are denoted as $x_k(t)$, $1 < k < 12$). The power coefficient is simply $x_0(t)$. The cepstrum vector and power streams are first normalized, and four feature vectors are derived in each frame by computing the first and second order differentials in time:

$$
\begin{aligned}
&X(t) = NormalizedCepstrumVector \\
&\Delta X(t) = X(t+2) - X(t-2), \Delta_l X(t) = X(t+4) - X(t-4) \\
&\Delta\Delta X(t) = \Delta X(t+1) - \Delta X(t-1)
\end{aligned}
\tag{2.1}
$$

$$
\begin{aligned}
&X_0(t) = x_0(t) : PowerStream, \\
&\Delta X_0(t) = X_0(t+2) - X_0(t-2), \\
&\Delta\Delta X_0(t) = \Delta X_0(t+1) - \Delta X_0(t-1). \text{ [85]}
\end{aligned}
$$

where the commas stand for the concatenation, resulting in four feature-vectors of 12, 24, 12, and 3 elements at every frame. Eventually, we use these vectors as the input to the ASR system.

2.2 Acoustic-Scoring: DNN, GMM, RNN

As the second step of an ASR system, the MFCC feature vectors are fed to an acoustic model, which generates the probability, a.k.a. acoustic-score, of seeing the different audio phonemes. Phonemes represent the very small sub-word units [97] which can be modeled by either an HMM or a WFST graph. This modeling simplifies the work of speech recognition as it shares the modeled sub-units across different words of the language. Context-sensitive phonemes are the norm, triphones [12] being the most common approach. A triphone is a particular phoneme that is combined with a particular predecessor and a particular successor. For instance, considering the phoneme AE in the words "man" and "lack", it sounds differently as in the former one it gets more nasal.

There are three basic approaches for evaluating the acoustic score of the different audio phonemes: GMM, DNN and RNN. In this thesis, we mainly focus on the DNN acoustic-scoring method. However, we have experimented our acceleration design under various speech datasets that are either DNN, GMM or RNN-based.

Regarding the GMM-based acoustic-scoring models, each triphone can be represented by its own separate weighted GMM model. However, computing such a big number of Gaussian functions is completely unfeasible for real-time speech recognition. In practice, multiple triphones sharing the same GMM are grouped into a cluster called as a senone [48], in order to reduce the computational cost of the acoustic model. The latest generic acoustic model has 5138 senones [98]. Furthermore, not all the senones are active for a given frame of speech, as some of them may be pruned away during the search process. Only the GMM for active senones are computed, as the acoustic likelihood for the other, inactive, senones is not required for the search [97].

The DNN is an alternative scheme for computing the acoustic scores for the speech phonemes. It has been proven that this model yields a better accuracy than the GMM-based ones. In this thesis, we rely on the DNN's architecture introduced in [116] and has been implemented in Kaldi [76], a well-known ASR toolkit. This network uses triphones (or context dependent phones) and, as in GMM solutions, they are clustered in senones. The DNN takes as input the set of MFCC features

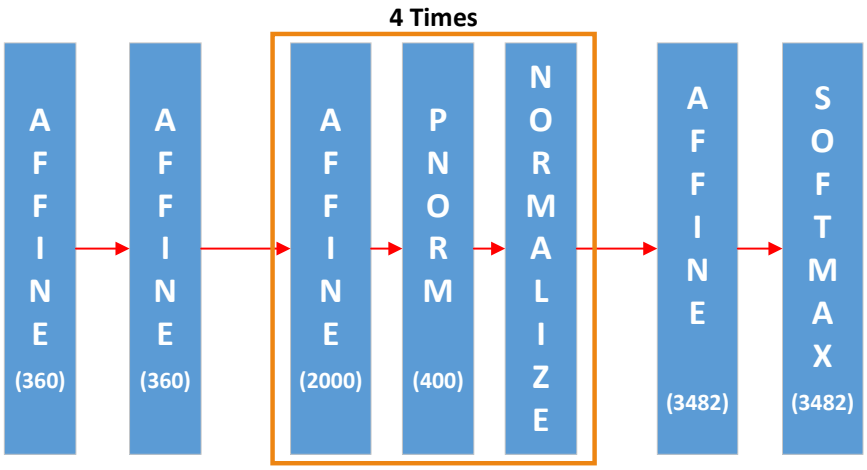


Figure 2.2: The structure of the Kaldi's DNN.

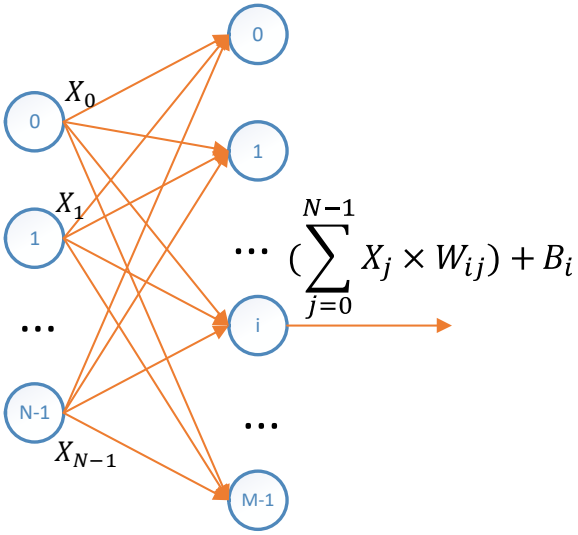


Figure 2.3: The Affine layer of the Kaldi's DNN.

describing a frame and it outputs the probabilities for each one of the senones for that particular frame.

Figure 2.2 shows the structure of the Kaldi's DNN. As illustrated, the network consists of several fully-connected (*Affine*) or maxout (*P-norm*, *Normalize*, or *Softmax*) layers [116]. Each layer is constructed using different number of neurons. First layer, called as the input layer, receives a 360-element feature vector that contains the features of the current frame and several adjacent frames, 4 previous and 4 next frames [75] (40 features for each frame). The *Affine* layer performs a matrix-vector multiplication on an input vector and the layer's weight matrix. Each column of the matrix contains the weights of a neuron. Figure 2.3 depicts the formulation for this layer. As illustrated, in order to compute the input to each neuron of the following layer, the corresponding row in the weight matrix is multiplied by the whole input vector plus adding a bias.

Regarding the maxout layers, a nonlinear function is applied to either one element or a group

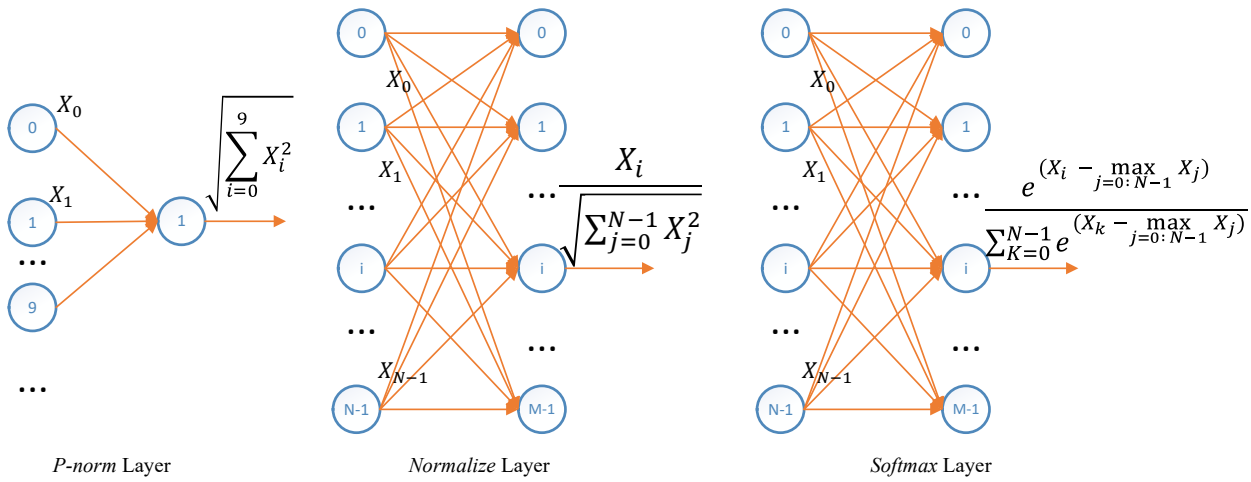


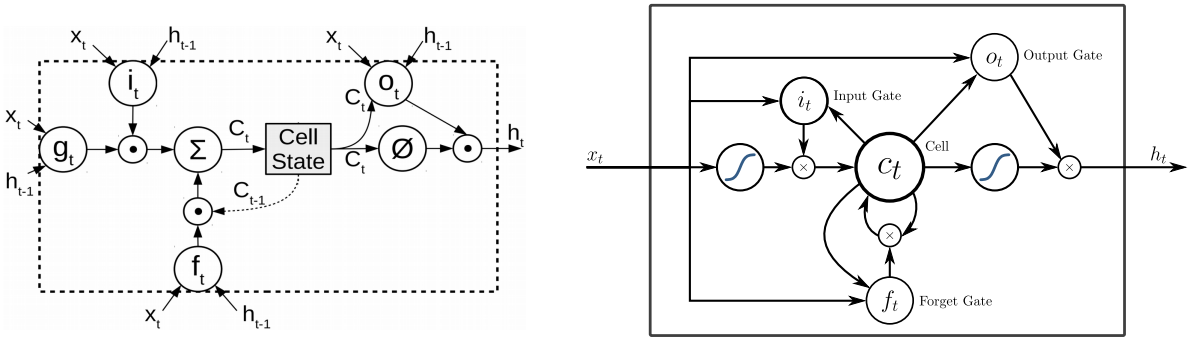
Figure 2.4: The three maxout layers of the Kaldi's DNN: P-norm, Normalize, and Softmax.

of input elements. Figure 2.4 illustrates the formulation for all these three types of layers. The *P-norm* layer, which is also used for image processing DNNs [16], reduces the number of outputs in the previous layer by a factor of 5. Each neuron takes as an input vector the outputs of 5 consecutive neurons in the previous layer and computes the length of this vector. The *Normalize* layer performs a normalization of the values computed in the previous layer. Finally, the *Softmax* layer, whose output dimension equals the number of context-dependent states, a.k.a senones, in the ASR system (3482 senones for Kaldi's network), computes the softmax function over the previous layer's outputs in order to normalize them into a probability distribution [10].

RNNs represent a well-known and effective alternative for sequence-to-sequence processing applications such as speech recognition. Long-Short-Term-Memory (LSTM) [45] and Gated-Recurrent-Unit (GRU) [23] are the two most commonly used RNN architectures. Figure 2.5 gives an overview of these two types of RNNs. As depicted, they rely on recurrent steps (through the backward links) in order to capture the past history inside the cell state and therefore obtain an accurate inference mechanism. However, this ability of keeping track of information for a long time comes with some intrinsic challenges of the data-dependencies for their computation. EESEN [63] is a popular speech RNN, which we use for our evaluation. This network consists of LSTM cells and contains 4 bi-directional layers. Unidirectional layers which include a forward pass only use past information, whereas bidirectional layers include both forward and backward sub-layers, that exploit both past and future context information.

2.3 Search-Engine: Viterbi Beam Search

By applying the acoustic likelihoods generated using an acoustic-scoring model, a search-engine called as the Viterbi beam search converts the sequence of phonemes' probabilities into a sequence of words. More specifically, the Viterbi search employs a WFST graph to find the sequence of output labels, or words, with maximum likelihood for the sequence of input labels, or phonemes, whose associated probabilities are generated by a DNN, GMM or RNN. The Viterbi search takes up the bulk of ASR's execution time when trying different types of acoustic-scoring models and,



(a) Long-Short-Term-Memory (LSTM) Unit (b) Gated Recurrent Unit (GRU)

Figure 2.5: The two most successful RNN architectures: LSTM and GRU.

therefore, is the main focus of this thesis. The complexity of the search process is due to the huge size of the recognition model employed to represent the characteristics of the speech [108, 109, 111].

WFST approach is the state-of-the-art in recognition networks for speech [66]. A WFST is a Mealy finite state machine that encodes a mapping between input and output labels associated with weights. In the case of speech recognition, the input labels represent the phonemes and the output labels the words. The WFST is constructed offline during the training process by using different knowledge sources such as context dependency of phonemes, pronunciation and grammar.

In order to run the Viterbi search on the speech graph(s), two different approaches have been proposed. The first scheme, that consists on using the fully-composed WFST, is common in both software [76] and hardware [52, 79, 111, 109]. In this approach, all the knowledge sources that are represented as an individual WFST are composed together to obtain a single WFST encompassing the entire speech dataset. Thus, the search simply explores one graph to decode the speech. However, as the offline-composed WFST normally requires more than 1 GB in practice for large vocabulary ASR systems, Viterbi search requires vast memory requirements as lots of hypotheses are expanded to explore the enormous search space.

As an alternative, in order to reduce the ASR's memory requirements, we can separate the speech WFSTs into several graphs that will be composed during the run-time of the Viterbi search algorithm. This scheme is known as the on-the-fly composition, which takes the two basic WFST-represented knowledge sources, called as Acoustic Model (AM) and Language Model (LM) [108], in order to decode the speech signal. AM represents the pronunciation of the different words in the vocabulary of a language, whereas LM scores the different hypotheses taking into account the probabilities of alternative sequences of words according to a given grammar. In this case, we perform the Viterbi search by dynamically composing the AM and LM, avoiding the generation of the fully-composed WFST.

In the following, we go through each one of these search-engine mechanisms by describing the organization of the different WFSTs. Moreover, we will explain how the Viterbi search algorithm works by using several simple examples.

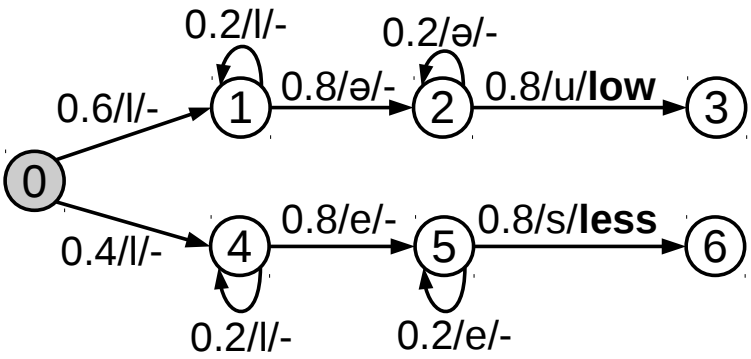


Figure 2.6: A simple WFST able to recognize the two words low and less.

Table 2.1: The acoustic likelihoods generated by the DNN according to the WFST of Figure 2.6, for an audio with three frames.

frame	l	ə	u	e	s
1	**0.9**	0.025	0.025	0.025	0.025
2	0.025	**0.7**	0.012	0.25	0.012
3	0.025	0.025	**0.9**	0.025	0.025

2.3.1 Viterbi Search Based on Fully-Composed WFST

The fully-composed approach has a major advantage over alternative representations of the speech model: it combines all the knowledge sources into one WFST, so the algorithm only needs to search over the resulting WFST without consideration of the knowledge sources. This characteristic is especially beneficial for a hardware implementation. Figure 2.6 shows a simple WFST for a very small vocabulary with two words. As illustrated, the WFST consists of a set of states and a set of arcs. Each arc has a source state, a destination state and three attributes: weight (or likelihood), phoneme (or input label) and word (or output label).

Viterbi search finds the word sequence with maximum likelihood, which is computed using a dynamic programming recurrence. The likelihood of the traversal process being in state j at frame f, $\psi_f(s_j)$, is computed from the likelihood in the preceding states as follows:

$$\psi_f(s_j) = \max_i \{\psi_{f-1}(s_i) \cdot w_{ij} \cdot b(O_f; m_k)\} \tag{2.2}$$

where w_{ij} is the weight of the arc from state i to state j, and $b(O_f; m_k)$ is the probability that the observation vector O_f corresponds to the phoneme m_k, i. e. the acoustic likelihood computed by the DNN. Table 2.1 shows the acoustic likelihoods generated by the DNN for the labels of the WFST shown in Figure 2.6, for an audio signal with three frames of speech. For instance, frame one has 90% probability of being phoneme l.

Figure 2.7 shows the trace of states expanded by the Viterbi search when using the WFST of Figure 2.6 and the acoustic likelihoods of Table 2.1. The search starts at state 0, the initial state of the WFST. Next, the Viterbi search traverses all possible arcs departing from state 0, considering the acoustic likelihoods of the first frame of speech to create new active states. For instance, the likelihood of being in state 1 at frame 1 is: $\psi_0(s_0) \cdot w_{01} \cdot b(O_1; l) = 1.0 \cdot 0.6 \cdot 0.9 = 0.54$. Note that

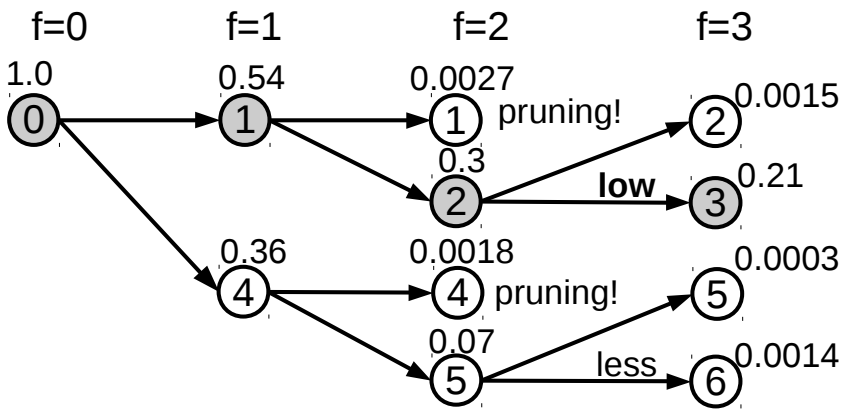


Figure 2.7: This figure shows a trace of the Viterbi algorithm when using the WFST in Figure 2.6 with the acoustic likelihoods shown in Table 2.1.

state 1 has only one predecessor in the previous frame. In case of multiple input arcs, all possible transitions from active states in the previous frame are considered to compute the likelihood of the new active state according to Equation 1. Therefore, the Viterbi search performs a reduction to compute the path with maximum likelihood to reach the state j at frame f. In addition to this likelihood, the algorithm also saves a pointer to the best predecessor for each active state, that will be used during backtracking to restore the best path.

For real WFSTs, it is unfeasible to expand all possible paths due to the huge search space. In practice, ASR systems employ pruning to discard the paths that are unlikely. In standard beam pruning, only active states that fall within a defined range, a.k.a. beam width, of the frame's best likelihood are expanded. In the example of Figure 2.7, we set the beam width to 0.25. With this beam, the threshold for frame 2 is 0.05: the result of subtracting the beam from the frame's best score (0.3). Active states 1 and 4 are pruned away as their likelihoods are smaller than the beam. The search algorithm combined with the pruning is commonly referred as Viterbi beam search [59].

On the other hand, representing likelihoods as floating point numbers between 0 and 1 might cause arithmetic underflow. To prevent this issue, ASR systems use log-space probabilities. Another benefit of working in log-space is that floating point multiplications are replaced by additions.

Regarding the arcs of the recognition network, real WFSTs typically include some arcs with no input label, a.k.a. epsilon arcs [113]. Epsilon arcs are not associated with any phoneme and they can be traversed during the search without consuming a new frame of speech. One of the reasons to include epsilon arcs is to model cross-word transitions. Epsilon arcs are less frequent than the arcs with input label, a.k.a. non-epsilon arcs. In Kaldi's WFST only 11.5% of the arcs are epsilon.

Note that there is a potential ambiguity in the use of the term state, as it might refer to the static WFST states (see Figure 2.6) or the dynamic Viterbi trace (see Figure 2.7). To clarify the terminology, in this thesis we use state to refer to a static state of the WFST, whereas we use token to refer to an active state dynamically created during the Viterbi search. A token is associated with a static WFST state, but it also includes the likelihood of the best path to reach the state at frame f and the pointer to the best predecessor for backtracking.

In spite of the simplicity of Viterbi expansion illustrated by the example of Figure 2.7, the search algorithm can get significantly complicated as there are typically more than 1000 frames to evaluate each speech utterance. Moreover, there is the need to evaluate thousands of hypotheses per frame, in order for the ASR system to achieve the desired level of accuracy (word-error-rate). For instance, considering the LibriSpeech [72] test set audio files, we observe that the average number of hypotheses per frame is larger than 20K, whereas the maximum number is bigger than 300K. By combining these two factors, we realize that the Viterbi search requires a vast memory subsystem since there are too many paths to explore at each frame as well as the high number of frames to decode at each utterance.

Furthermore, as the WFST is enormously large and tends to get bigger by increasing the complexity of the recent highly-accurate speech models, Viterbi search becomes the main memory bottleneck due to its unpredictable behaviour toward the hypotheses generation. By using the general-purpose solutions, there is no optimum solution to adapt the memory hierarchy system in order to perform Viterbi search fast and energy-efficient at the same time. This is due to two facts: first, there is no trivial solution for bringing the data necessary for the Viterbi expansion in advance in order to keep the processor's pipeline busy because of the large number of cache misses; second, due to the pruning mechanism for the Viterbi beam search algorithm, its dataflow becomes more complex as it brings up so many divergences when dealing with the branches and the memory accesses during the search.

Considering the best solution implemented on a high-end desktop GPU which shares the Viterbi search between many parallel workers (threads and streaming multiprocessors), we have seen that there is the high power demand of over 100 W, since there is so many memory requests initiated by the parallel cores of the GPU in order to efficiently hide the memory latency with the decoding process.

2.3.2 Viterbi Search Based on On-the-fly WFST Composition

On-the-fly composition [18, 17, 46] represents an alternative implementation that performs the Viterbi search by dynamically composing the AM and LM graphs, avoiding the generation of the fully-composed WFST. AM and LM can be efficiently encoded as a WFST. AM represents the pronunciation of the different words in the vocabulary of a language, whereas LM scores the different hypotheses taking into account the probabilities of alternative sequences of words according to a given grammar.

There are mainly two different strategies proposed for on-the-fly decoders: one-pass [46, 18] and two-pass [60] search methods. In the one-pass strategy, the search is done at the same time of composing the sub-WFSTs on-the-fly as necessary. In the two-pass approach, these phases are separated in a way that AM is searched first to produce multiple hypotheses and a word lattice, which is then rescored using the LM. As the rescoring phase of the two-pass method cannot be executed until the end of AM search, it typically leads to larger latencies that are harmful for real-time ASR decoders. In this paper, we selected the one-pass approach for a hardware implementation with the aim of achieving decoding times similar to the traditional fully-composed decoders.

An AM WFST uses the acoustic scores generated by either GMM, DNN or RNN to identify

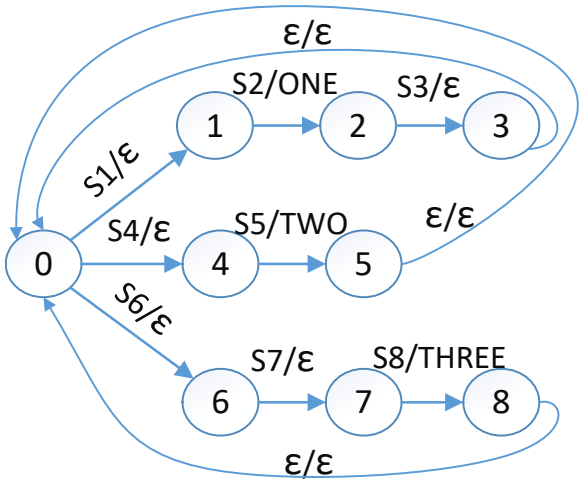


Figure 2.8: An Acoustic Model (AM) which detects 3 words: ONE, TWO and THREE.

words and their corresponding probabilities given the audio signal. Then, it outputs the different sequences of words, a.k.a. the hypotheses that best match the identified phonemes and their respective probability. Figure 2.8 shows an example of a simple AM graph for a three-word vocabulary. The input label of an arc is the phoneme ID (Sx) of the language, whereas the output label contains the word ID. Epsilon output label means there is no word ending in that arc, whereas epsilon input label means that the arc can be traversed without consuming any acoustic score. Arcs with non-epsilon output label, i.e. arcs associated with a word ID, are also known as cross-word transitions. Figure 2.9 shows an example of a simple LM. The input and output label of an arc in the LM contain the same word ID, whereas the weight is the likelihood of the corresponding unigram (1-word), bigram (2-word) or trigram(3-word) grammar models. Each state is associated with a given word history. State 0 is the initial state with empty history, the arcs departing from state 0 represent the unigram likelihoods. States 1, 2 and 3 have a history of one word, arcs departing from these states encode the bigram likelihoods. Finally, states 4, 5 and 6 have a history of two words, arcs departing from these sates represent the trigram likelihoods. For example, being at state 6 means that the last two input words were $THREE$. TWO, if next word is ONE then we transition to state 5 to update the history, applying the trigram probability of $Prob(ONE \mid THREE\ .\ TWO)$ to rescore this hypothesis.

For large vocabulary systems, it is unfeasible to include arcs for all the possible combinations of two and three words. Combinations whose likelihood is smaller than a threshold are pruned to keep the size of the LM manageable. Back-off arcs are then added to each state of the WFST in order to recover for the cases that a pruned combination of words in the grammar is observed at the input hypothesis. If a combination of words is not available, then the WFST uses these extra arcs to back off from trigram to bigram, or from bigram to unigram. Furthermore, the back-off arcs are associated with a weight to reduce the likelihood of these hypotheses.

Figure 2.10 illustrates how the Viterbi search does the decoding with the on-the-fly scheme. Each node in the search graph, a.k.a. token, is associated with a state in the AM graph and a state in the LM. For instance, token (2, 1) indicates that we are at state 2 of AM and state 1 of LM. The search starts at the initial state in both WFSTs, i.e. token (0, 0), and proceeds as

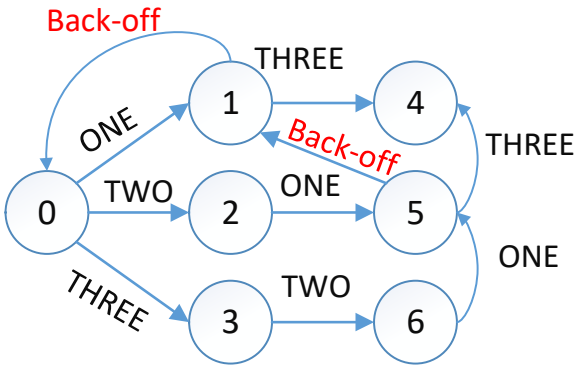


Figure 2.9: A 3-gram Language Model (LM) of the AM of Figure 2.8.

follows. In the first frame, the arcs of state 0 in AM model are traversed to create new tokens for the next frame of speech. The likelihoods of these new tokens are computed by using the likelihood of the source token and the acoustic scores of the phonemes associated with the arcs (S1, S4, S6), provided by GMM/DNN/RNN. Since these AM arcs do not correspond to cross-word transitions, i.e. they have epsilon output label, no arc is traversed in the LM and the new tokens remain in state 0 in the second WFST. The same process is repeated in the next frame, but in this case the search traverses AM arcs with non-epsilon output labels, i.e. words ONE and TWO are recognized in different hypotheses.

When a cross-word transition is traversed in the AM model, its word ID is used to transition in the LM graph. The weight of the LM arc is also employed to compute the likelihood of the new token, in order to modulate the score of the hypothesis according to the grammar. Therefore, the AM graph drives the search, whereas the LM graph is used to rescore the hypotheses when an arc with a word ID, i.e. a cross-word transition, is traversed in the AM graph. Due to the large search space, pruning of the search graph is also applied to discard unlikely hypotheses.

Even though the process of on-the-fly composition can be performed in one step if the AM and LM models are composed offline [66], it requires a multiplicative combination of states and arcs in the AM and LM graphs, resulting in large WFST models. Table 2.2 shows the sizes of these graphs for several ASR decoders. As it can be seen, the AM and LM models have sizes smaller than 102 Megabytes, whereas the composed WFST exceeds one Gigabyte for some decoders.

With the on-the-fly mechanism, there is no need to store every information of the fully-composed WFST while decoding the speech. The WFSTs of the AM and LM are composed on demand as required by the Viterbi search. Note that the search graph of Figure 2.10 represents the different alternative interpretations of the speech generated by the Viterbi search. The search graph is required for both offline and on-the-fly composition, and its memory footprint is fairly small compared to the size of the WFSTs. Therefore, the storage requirements are reduced by an order of magnitude (see Table 2.2), as only the individual AM and LM are stored in main memory.

A main drawback of on-the-fly composition is that it increases the complexity of the decoder since it needs to fetch states and arcs from two WFSTs. Furthermore, it increases the amount of computations, since the LM likelihoods are applied at runtime, whereas they are merged offline with AM information in the fully-composed approach.

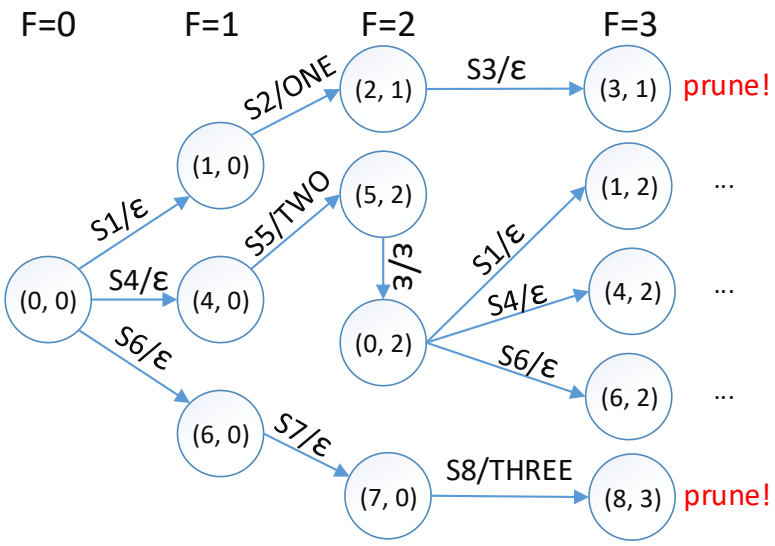


Figure 2.10: A partially dynamic-composed search graph using the AM and LM of Figures 2.8 and 2.9.

Table 2.2: Size of the individual AM and LM graphs and the fully-composed WFST in Megabytes.

ASR Decoder	AM WFST	LM WFST	Composed WFST
Kaldi-TEDLIUM	33	66	1090
Kalid-Librispeech	40	59	496
Kalid-Voxforge	2.8	2.3	37
EESEN-TEDLIUM	34	102	1226

The other important drawback is the cost of fetching the arcs in the LM graph. The arcs for a given state can be easily located both in the AM graph and the fully-composed WFST, as it only requires one indirection to fetch the state's information that provides the address of the first arc and the number of arcs. Note that when a state is expanded, all its arcs are explored sequentially to generate new tokens. However, when a cross-word transition is traversed in the AM model, the search has to traverse the arc associated with the corresponding word ID in the LM, i.e. the arc whose input label matches that word ID. Locating the correct arc requires some sort of search across the arcs of a given state, which might be expensive for two reasons. First, states in the LM have thousands of arcs. Second, due to the aforementioned back-off mechanism, there might be no arc with the target word ID for a given state and, in this case, the search would be repeated multiple times for different states in the LM. Implementing the location of the arc as a linear search increases the execution time by 10x with respect to offline composition. Alternatively, sorting the outgoing arcs of each state by word ID to use binary search reduces the slowdown to 3x, which still represents a huge overhead, making the fetching of arcs in the LM the most critical operation for on-the-fly composition. In Section 3, we propose a specialized hardware to speed up this operation and locate the arcs in the LM graph with a very small overhead.

3

Experimental Methodology

In this chapter, we review the methodology employed throughout this thesis in order to evaluate the different architectures, with special emphasis in the methods employed to assess performance, area, power and energy-consumption. We also describe the tools used for the evaluation of our speech recognition accelerator.

The system platform that we rely on for all our experiment is a heterogeneous architecture, as shown in Figure 3.1, containing a CPU, a GPU, a Viterbi accelerator and a DNN ASIC. These four modules are connected together through a shared memory. Regarding all our benchmarks, we consider CPU for running the feature-extraction part of the ASR pipeline. For the acoustic-scoring, we either use the GPU or DNN accelerator at the each step of our proposals and based on the types of the model that we are applying. Finally, the Viterbi accelerator, which is the main proposal of this thesis, is responsible for performing the graph search and managing the WFST datasets. Moreover, each accelerator is equipped with its own memory controller in order to communicate with the memory subsystem of the SoC architecture through the efficient PCI express 4.0 Bus protocol.

In the following sections, we firstly define the methodology we take for developing the accelerator's architecture, starting form the hardware simulation to the Register-Transfer-Level (RTL) modeling and gate-level synthesis. Secondly, we describe the tools we use in order to measure the performance and power usage of the CPU and GPU systems. Finally, we introduce the speech datasets which we use for this thesis experiments and evaluations.

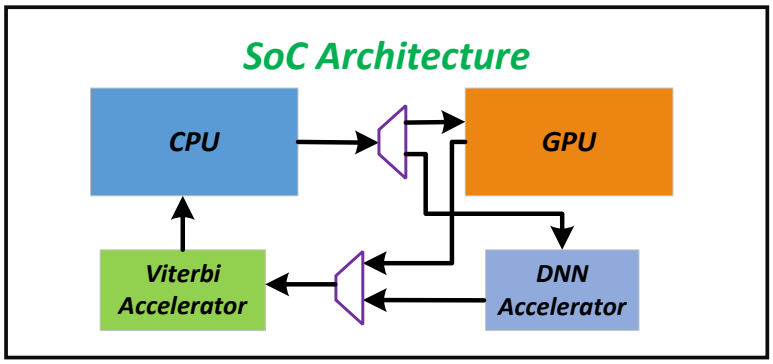


Figure 3.1: The ASR SoC architecture. It includes a CPU, a GPU, and Viterbi and DNN accelerators. The CPU processes the input audio signal in order to extract the MFCC features. Next, either the GPU or DNN accelerator evaluates the acoustic scores of the different speech phonemes. Then, Viterbi accelerator performs the Viterbi search in order to decode the speech. Finally, CPU gets the sequence of words by running a backtracking phase.

3.1 Hardware Acceleration Modeling and Evaluation

Figure 3.2 illustrates the different stages we take in order to develop a hardware acceleration design, from the top-level considering the application's algorithm to the low-level of getting a synthesized layout for the hardware. As the first phase of accelerator's design, we partition each part of the ASR's pipeline that we want to accelerate, i.e. DNN and Viterbi search, into several tasks. Next, as shown by Figure 3.2, we can have a Control-data Dependency Graph (CDG) which shows the interaction between the different tasks of the algorithm we are modeling the hardware accelerator for. Then, we assign each task to a hardware component and by joining them, we generate a pipeline that performs either Viterbi expansion or DNN acoustic-scoring. In order to evaluate the functionality of this architecture, we model the pipeline with all the components' specification in a simulator. Moreover, the simulator is cycle-accurate, meaning that it counts the total execution cycles considering the delay of each component's operations.

We build the simulator using several C++ primitive classes that accurately mimic the hardware behaviour, such as the clock signal (*posedge* and *negedge*), and input and output ports. Furthermore, we use a generic *Component* class that all the pipeline stages are inherited from. At each component, we certify that all the input signals are ready for process at the *negedge* of the clock, whereas the output signals are only written at the *posedge* of the clock. By using a static *Simulator* class we register all the pipeline components whose input and output ports are correctly linked with each other. The *Simulator* class then runs the pipeline by calling all the components' *posedge* and *negedge* functions, respectively. It also keeps track of the soonest next cycle for which a component needs to operate in order to save simulation time.

Not only is our developed simulator able to perform the function of the accelerated part of ASR, but it also gives the execution cycles plus the activity information of each accelerator's component. However, in order to estimate the main design parameters such as latency, area and power dissipation of the accelerator, as depicted by Figure 3.2, we need to implement the pipeline stages in hardware. To do so, we first implement each component's functionality in Verilog using its RTL description. Second, by synthesizing each module with the Synopsys Design Compiler [5] using

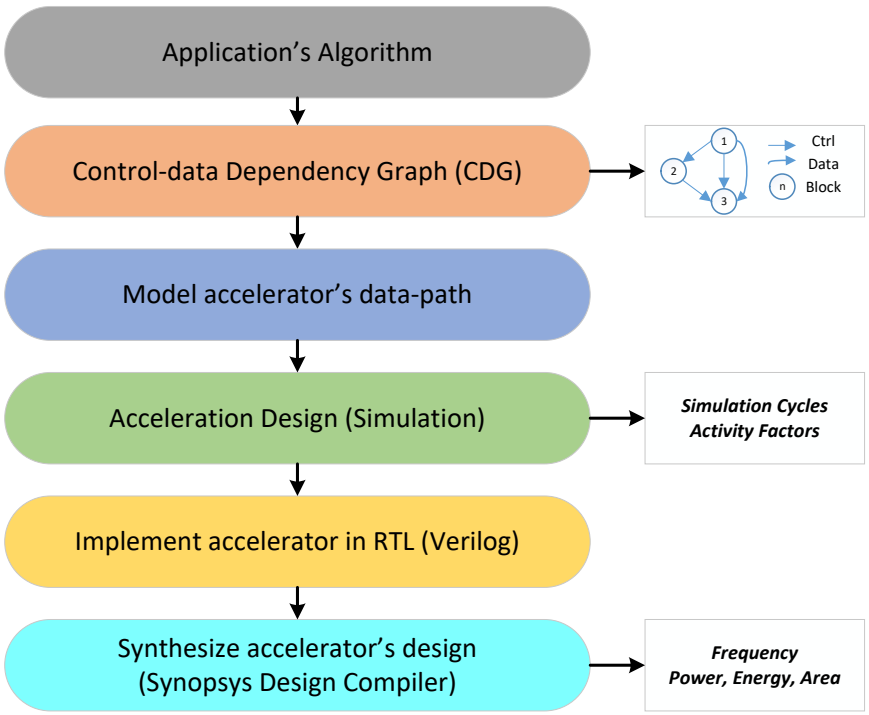


Figure 3.2: The top-down methodology of designing the hardware accelerator. We start form the application's algorithm. Next, by building the CDG, we divide the algorithm into several tasks, which are then modeled by the hardware. Then, by developing a simulator we verify the functionality of the hardware and, in addition, we extract the execution cycles and activity factors. Finally, by implementing the hardware accelerator in RTL using Verilog/VHDL, we can measure the frequency, power, energy and area of the design by synthesizing that using the Synopsys Desing Compiler.

a 32nm technology library, we can generate the gate-level netlist. Design Compiler also reports the critical-path delay and the static and dynamic power dissipation. Regarding the dynamic power, it considers a default 50% activity factor for all the signals in the netlist.

Furthermore, in order to get the power, area and delay of the memory components of the accelerator, such as caches, buffers and scratch-pad memories, we use CACTI [67]. We employ energy-delay as the optimization function for CACTI's exploration. To set the frequency of the accelerator, we consider the critical-path-delay and access-time reported by the Design Compiler and CACTI, respectively. Then, we take the maximum delay among the different components, in order to set it as the cycle time-period. Moreover, we model the off-chip DRAM main memory by using the Micron power model for an 8-GB LPDDR4 [100, 101].

Regarding the total power and energy estimation, we combine all the measurements of the tools together. The cycle-accurate simulator provides the activity factors for the different components and the total cycle count. Then, by having the frequency set using the latency evaluation of CACTI and Design Compiler, we can compute the execution time and static energy. In addition, we measure the dynamic energy by combining the simulator's activity factors with the estimations of the Design Compiler, CACTI and Micron power models.

Table 3.1: CPU parameters.

CPU	Intel Core i7 6700K
Number of cores	4
Technology	14 nm
Frequency	4.2 GHz
L1, L2, L3	64 KB, 256 KB per core, 8 MB

Table 3.2: High-end GPU parameters.

GPU	NVIDIA GeForce GTX 980
Streaming multiprocessors	16 (2048 threads/multiprocessor)
Technology	28 nm
Frequency	1.28 GHz
L1, L2 caches	48 KB, 2 MB

Table 3.3: Mobile GPU parameters.

GPU	Nvidia Tegra X1
Streaming multiprocessors	2 (2,048 threads/multiprocessor)
Technology	20 nm
Frequency	1.0 GHz
Level-2 cache	256 Kbytes

3.2 CPU and GPU Implementations

Regarding the CPU, we use the software implementation of the Viterbi beam search and the DNN and GMM acoustic-scoring included in Kaldi [76]. Moreover, we use EESEN [63] for the end-to-end RNN-based ASR systems. We measure the performance of the software implementation on a real CPU with the parameters shown in Table 3.1. We employ Intel RAPL library [104] to measure energy consumption. We use GCC 4.8.4 to compile the software using -O3 optimization flag.

Regarding GPU, we run state-of-the-art CUDA implementations of the different ASR decoders on both a high-end and a mobile GPU. For the Viterbi search, we use the state-of-the-art GPU version presented in [89]. For the GMM and the DNN, we use the implementation provided in Kaldi [76]. For the RNN, we use the GPU-accelerated version included in EESEN [63]. We apply the nvcc compiler included in CUDA toolkit version 7.5 with -O3 optimization flag. We run our CUDA implementation on a high-end GPU (Geforce GTX 980) and a low-power mobile GPU (Jetson Tegra-X1) with the parameters shown in Tables 3.2 and 3.3, respectively. To measure the energy consumption of the mobile GPU, we read the registers of the Texas Instruments INA3221 power monitor included in the Jetson TX1 platform, which provides the actual energy by monitoring the GPU power rail as described in [31]. Moreover, we use the NVIDIA Visual Profiler [71] to obtain performance and power of the high-end GPU.

3.3 Speech Datasets

Initially, we employed the pre-trained WFST for the English language provided in the Kaldi toolset [76], that is based on the proprietary Fisher English corpus [25], for evaluating the Viterbi accelerator. Then, we leveraged several openly available speech corpora, since evaluation of some of the techniques required access to the full training datasets. More specifically, we employ Librispeech [72] dataset, that contains more than 900 hours of speech from audio books. Speech models for Librispeech are trained with a large vocabulary of 200K words. Furthermore, we use Tedlium [88] datasets, that is collected from TED talks and is more challenging as it consists of spontaneous speech in a noisy environment. WFSTs for Tedlium are trained with a vocabulary of 150K words. Finally, we also use Voxforge [27] datasets to train speech models with a vocabulary of 13.9K words. On the other hand, the speech models for these datasets are based on different approaches for the acoustic-scoring, either Gaussian Mixture Model (GMM) (Tedlium and Voxforge) or DNN (Librispeech and Fisher English datasets). Overall, our system is evaluated for thousands of hours of speech and hundreds of different speakers with different accents and conditions, including both clean and noisy audio.

4

Viterbi Search Accelerator

In this chapter we describe our high-performance, low-power, hardware-accelerated SoC for speech recognition. In this system, we propose a Viterbi accelerator besides the use of a CPU and a GPU for the rest of ASR pipeline. We first provide an overview of our system. Then, we elaborate on the architecture of the accelerator. Moreover, we explain all the optimization techniques we applied in order to remove the accelerator's main bottlenecks. After that, we introduce a low-power technique which removes a large portion of the accelerator's power dissipation by reducing the leakage power. Finally, we show some experimental results and a comparison with the previous proposals.

4.1 Overall Accelerated ASR System

Our ASR system is based on the hybrid approach that combines a DNN for acoustic scoring with the Viterbi beam search, which is the state-of-the-art scheme in speech recognition. DNN evaluation is easy to parallelize and, hence, it achieves high performance and energy-efficiency on a GPU. On the other hand, the Viterbi search is the main bottleneck and is hard to parallelize, so we propose a dedicated hardware accelerator specifically tailored to the needs of Viterbi beam search algorithm. Therefore, our overall ASR system combines the GPU, for DNN evaluation, with an accelerator for Viterbi search.

Figure 4.1 shows our overall ASR system for both mobile and desktop platforms. The CPU performs feature extraction to generate the audio frames in main memory. The GPU computes the DNN for those audio frames and generates the acoustic likelihoods. The accelerator performs the Viterbi search by using the acoustic scores in order to generate the word lattice in system memory. Finally, the CPU performs the backtracking to obtain the most likely sequence of words.

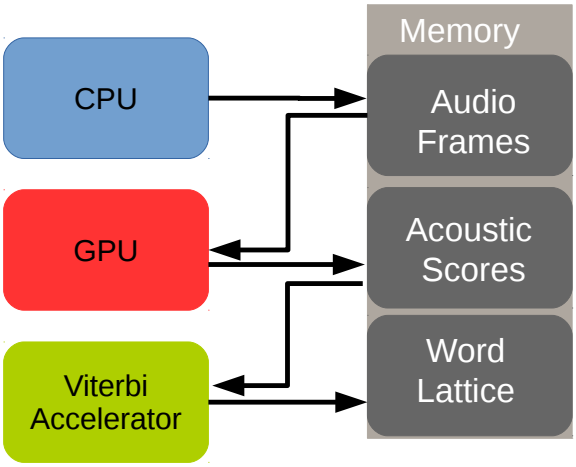


Figure 4.1: The architecture of our proposed accelerated design for ASR system.

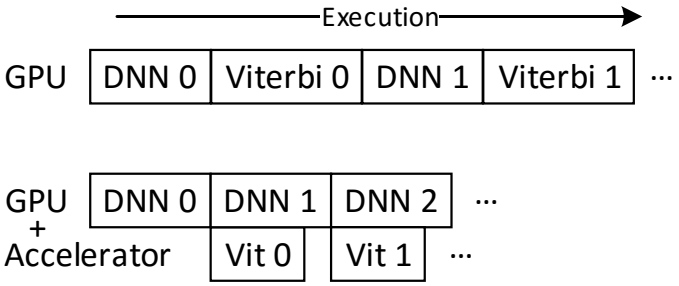


Figure 4.2: Execution of the GPU-only ASR system (top) and our proposed architecture combining a GPU and the Viterbi accelerator (bottom).

In our ASR system, the input frames of speech are grouped in batches. The accelerator and the GPU work in parallel, processing different batches in a pipelined manner: the GPU computes the DNN for the current batch while the accelerator performs Viterbi search for the previous batch. Note that the results computed by the GPU, i.e. the acoustic scores for one batch, have to be transferred from GPU memory to accelerator's memory. In order to hide the latency required for transferring acoustic scores, a double-buffered scratch-pad memory is included in the accelerator, called as Acoustic Likelihood Buffer. This memory stores the acoustic scores for the current and the next frame of speech. The accelerator decodes the current frame of speech while the acoustic scores for the next frame are fetched from memory, overlapping computations with memory accesses.

In comparison with a GPU-only architecture, we can better overlap the computation of the ASR pipeline stages. Figure 4.2 shows a part of the execution both on our proposed design and a GPU. As we can see, while the GPU is busy with the current batch, Viterbi accelerator is operating on the previous one. Therefore, we can save a lot of time and energy since the GPU, as a power-hungry unit, only does DNN by efficiently parallelizing its computation, while the Viterbi accelerator is customized to run the search that is the main bottleneck of ASR.

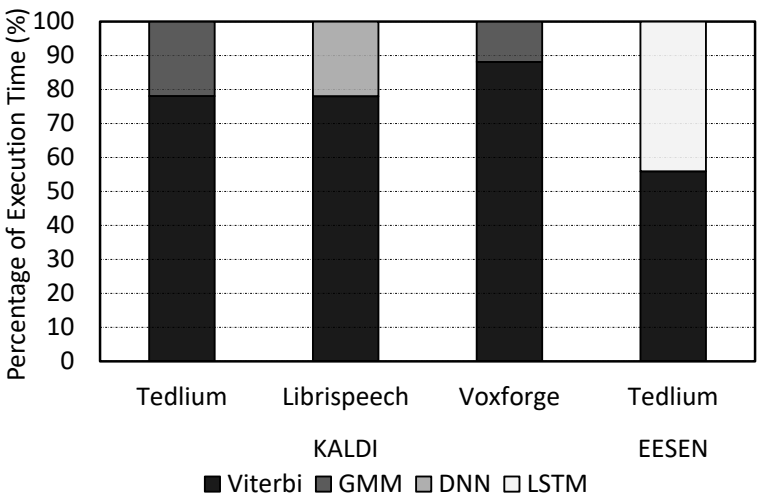


Figure 4.3: Percentage of execution time for different com- ponents of Kaldi and EESEN speech decoders. Measured on NVIDIA Tegra X1 mobile SoC.

4.2 Viterbi Accelerator

In this section we provide a detailed description of the Viterbi accelerator. As aforementioned, the accelerator focuses on the Viterbi beam search as it represents the vast majority of the execution time in all the analyzed platforms. Furthermore, we evaluate several different ASR decoders to make a strong case for our Viterbi search accelerator. Figure 4.3 shows the percentage of execution time that ASR's pipeline stages take in Kaldi and EESEN when running on a mobile GPU. As it can be seen, the Viterbi search is by far the main bottleneck, taking more than 78% of execution time in Kaldi for both the GMM- and DNN-based models. On the other hand, for the RNN-based speech decoder, the Viterbi search represents the main bottleneck taking more than 55% of execution time.

Figure 4.4 illustrates the architecture of the accelerator which consists of a pipeline with five stages. In addition to a number of functional blocks for performing the graph processing steps, it includes several on-chip memories to speed up the access to different types of data required by the search process. More specifically, the accelerator includes three caches (State, Arc and Token), two hash tables to store the tokens for the current and next frames of the speech, and a scratchpad memory to store the acoustic likelihoods computed by GPU executing the DNN.

Regarding the speech graph, we use the fully-composed Fisher English WFST produced by merging all the knowledge resources offline. This WFST which is also used in the experiments of this work [76] has a large vocabulary of 125k words and it contains more than 13M states and more than 34M arcs. Thus, the accelerator cannot store the WFST on-chip due to its huge size, that is 682 MBytes in total. Regarding the representation of the WFST, we use the memory layout proposed in [21]. In this layout, states and arcs are stored separately in two different arrays. For each state, three attributes are stored in main memory: index of the first arc (32 bits), number of non-epsilon arcs (16 bits) and number of epsilon arcs (16 bits) packed in a 64-bit structure. All the outgoing arcs for a given state are stored in consecutive memory locations in the array of arcs; the non-epsilon arcs are stored first followed by the epsilon arcs. For each arc, four attributes are stored packed in 128 bits: index of the arc's destination state, transition weight, input label (phoneme id)

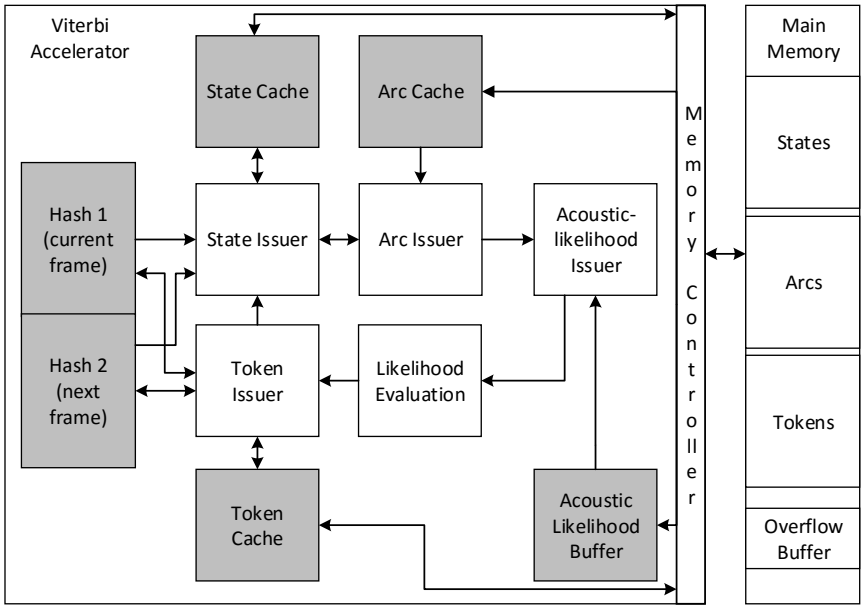


Figure 4.4: The architecture of the Viterbi accelerator for speech recognition.

and output label (word id), each represented as a 32-bit value. The State and Arc caches speed up the access to the array of states and arcs respectively.

On the other hand, the accelerator also keeps track of the tokens generated dynamically throughout the search. Token's data is split into two parts, depending on whether the data has to be kept until the end of the search or it is only required for a given frame of speech: a) the back-pointer to the best predecessor is required for the backtracking step to restore the best path when the search finishes, so these data are stored in main memory and cached in the Token cache; b) the state index and the likelihood of reaching the token are stored in the hash table and are discarded when the processing of the current frame finishes.

Due to the pruning described in Section 2.3.1, only a very small subset of the states are active at any given frame of speech. We empirically found that only around 25K tokens (out of 34M) are explored at each frame of speech. The accelerator employs two hash tables to keep track of the active states, or tokens, created dynamically for the current and next frames respectively. By using a hash table, the accelerator is able to quickly find out whether the WFST's state has already been visited in a frame of speech.

Regarding the ASIC configuration of the Viterbi accelerator, we use the parameters shown in Table 4.1. We have done an extensive design space exploration in order to set the sizes of the on-chip buffers, such as caches, and hash tables. In the following, we will present some of these exploration results.

Table 4.1: Hardware parameters for the accelerator.

Technology	28 nm
Frequency	600 MHz
State Cache	512 KB, 4-way, 64 bytes/line
Arc Cache	1 MB, 4-way, 64 bytes/line
Token Cache	512 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 states
Arc Issuer	8 in-flight arcs
Token Issuer	32 in-flight tokens
Acoustic Likelihood Issuer	1 in-flight arc
Likelihood Evaluation Unit	4 fp adders, 2 fp comparators

4.2.1 Accelerator's Pipeline

The Viterbi accelerator, illustrated in Figure 4.4, implements the Viterbi beam search algorithm described in Section 2.3.1. We use the example shown in Figure 2.7 to illustrate the behavior of the accelerator. More specifically, we describe how the hardware expands the arcs of frame 2 to generate the new tokens in frame 3. Initially, the hash table for the current frame (Hash 1) stores the information for tokens 1, 2, 4 and 5. The State Issuer fetches these tokens from the hash table, including the index of the associated WFST state and the likelihood of reaching the token. The State Issuer then performs the pruning using a threshold of 0.05: the likelihood of the best token (0.3) minus the beam (0.25). Tokens 1 and 4 are pruned away as their likelihoods are smaller than the threshold, whereas tokens 2 and 5 are considered for arc expansion. Next, the State Issuer generates memory requests to fetch the information of WFST states 2 and 5 from main memory, using the State Cache to speed up this process. The state information includes the index of the first arc and the number of arcs. Once the State Cache serves these data, they are forwarded to the Arc Issuer.

The Arc Issuer generates memory requests to fetch the outgoing arcs for states 2 and 5, using the Arc Cache to speed up the arc fetching. The arc information includes the index of the destination state, weight, phoneme index and word index. For example, the second arc of state 2 has destination state 3, weight equal to 0.8 and it is associated with phoneme u and word low. Once the Arc Cache provides the data of an arc, it is forwarded to the next pipeline stage. The Acoustic Likelihood Issuer employs the phoneme's index to fetch the corresponding acoustic score computed by the DNN. A local on-chip memory, the Acoustic Likelihood Buffer, is used to store all the acoustic scores for the current frame of speech.

In the next pipeline stage, i.e. the Likelihood Evaluation, all the data required to compute the likelihood of the new token according to Equation 2.2 is already available: the likelihood of reaching the source token, the weight of the arc and the acoustic score from the DNN. The Likelihood Evaluation performs the summation of these three values for every arc to compute the likelihoods of reaching the new tokens in the next frame. Note that additions are used instead of

multiplications as the accelerator works in log-space. In the example of Figure 2.7, the second arc of token 2 generates a new token in the next frame, token 3, with likelihood 0.21: the likelihood of the source token is 0.3, the weight of the arc is 0.8 (see Figure 2.6) and the acoustic score for phoneme u in frame 3 is 0.9 (see Table 2.1).

Finally, Token Issuer stores the information for the new tokens in the hash table for the next frame (Hash 2). In this example, it saves the new tokens for states 2, 3, 5 and 6 together with their associated likelihoods in the hash table. In addition, the index of the source token and the word index are stored in main memory, using the Token Cache, as this information is required for backtracking in order to recover the best path. Note that different arcs might have the same destination state. In that case, Token Issuer only saves the best path for reaching the destination state, i. e. the path with maximum likelihood among all the different ingoing arcs.

The hash table is accessed using the state index. Each entry in the hash table stores the likelihood of the token and the address where the back-pointer for the token is stored in main memory. In addition, each entry contains a pointer to the next active entry, so all the tokens are in a single linked list that can be traversed by the State Issuer in the next frame. The hash table includes a backup buffer to handle collisions (states whose hash function maps them to the same entry). In case of a collision, the new state index is stored in a backup buffer, and a pointer to that location is saved in the original entry of the hash. Each new collision is stored in the backup buffer and linked to the last collision of the same entry, all collisions forming a single linked list.

On the other hand, in case of an overflow in a hash table, the accelerator employs a buffer in main memory, labeled as Overflow Buffer in Figure 4.4, as an extension of the backup buffer. Overflows significantly increase the latency to access the hash table, but we found that they are extremely rare for common hash table sizes.

The Acoustic Likelihood Buffer contains the likelihoods computed by the DNN. In our system, the DNN is evaluated in the GPU and the result is transferred to the aforementioned buffer in the accelerator. The buffer contains storage for two frames of speech. The accelerator expands the tokens for the current frame while it fetches the acoustic scores for the next frame, overlapping computations with memory accesses.

The result generated by the accelerator is a dynamic trace of tokens in main memory, such as the trace of Figure 2.7), together with the address of the best token in the last frame. The backtracking is done on the CPU, following backpointers to get the sequence of words in the best path. The execution time of the backtracking is negligible compared to the Viterbi search so it does not require hardware acceleration.

4.2.2 Analysis of Caches and Hash Tables

Misses in the caches and collisions in the hash tables are the only sources of pipeline stalls and, therefore, the parameters for those components are critical for the performance of the overall accelerator. In this section, we evaluate different configurations of the accelerator to find appropriate values for the capacity of the State, Arc and Token caches and the hash tables.

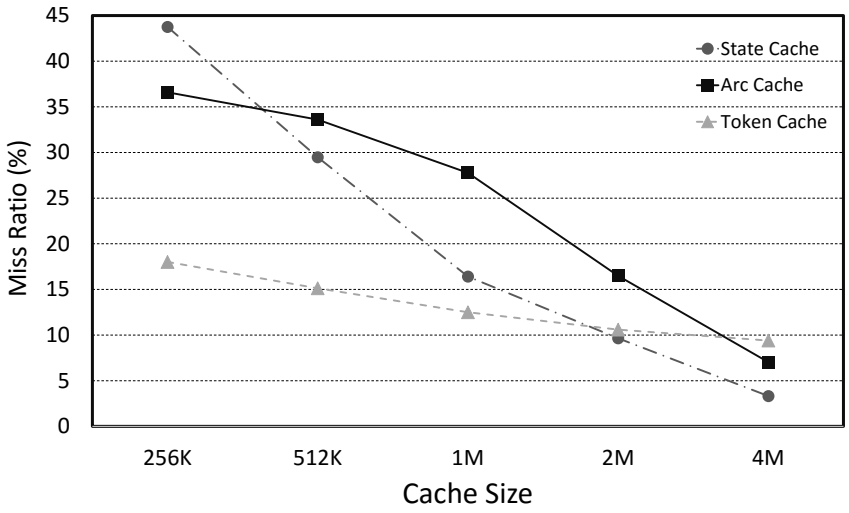


Figure 4.5: Evaluation of the miss-ratio vs capacity for the different caches in the Viterbi accelerator.

Figure 4.5 shows the miss ratios of the caches for different capacities. As it can be seen, even large capacities of 1-2 MBytes exhibit significant miss ratios. These large miss ratios are due to the huge size of the datasets, mainly the arcs and states in the WFST, and the poor spatial and temporal locality that the memory accesses to those datasets exhibit. Only a very small subset of the total arcs and states are accessed on a frame of speech, and this subset is sparsely distributed in the memory. The access to the array of tokens exhibits better spatial locality, as most of the tokens are added at the end of the array at consecutive memory locations. For this reason, the Token cache exhibits lower miss ratios than the other two caches for small capacities of 256KB-512KB.

Large cache sizes can significantly reduce miss ratio, but they come at a significant cost in area and power. Furthermore, they increase the latency for accessing the cache, which in turn increases total execution time. For our baseline configuration, we have selected 512 KB, 1 MB and 512 KB as the sizes for the State, Arc and Token caches respectively. Although larger values provide smaller miss ratios, we propose to use instead other more cost-effective techniques for improving the memory subsystem, described in Section 4.3.

Regarding the hash tables, Figure 4.6 shows the average number of cycles per request versus the number of entries in the table. If there is no collision, requests take just one cycle, but in case of a collision, the hardware has to locate the state index by traversing a single linked list of entries, which may take multiple cycles (many more if it has to access the Overflow Buffer in main memory). For 32K-64K entries, the number of cycles per request is close to one. Furthermore, the additional increase in performance from 32K to 64K is very small as it can be seen in the speedup reported in Figure 5. Therefore, we use 32K entries for our baseline configuration which requires a total storage of 768 KBytes for each hash table, similar to the capacities employed for the caches of the accelerator.

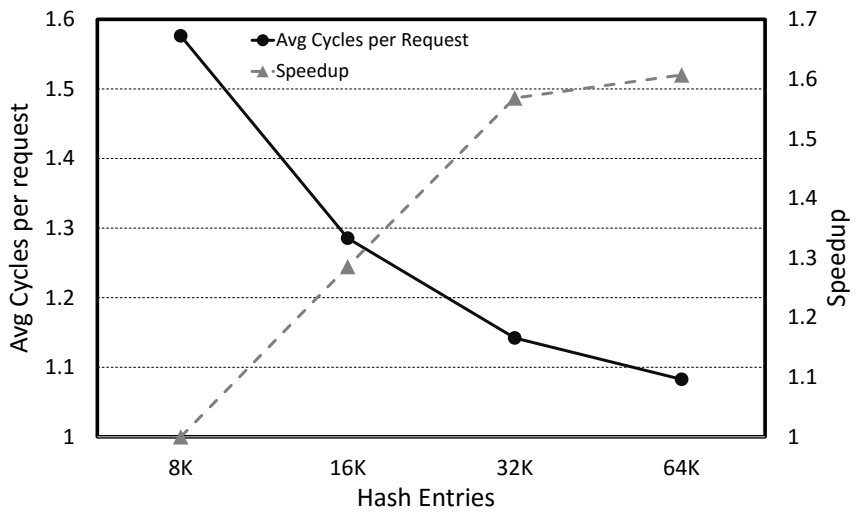


Figure 4.6: The average cycles per request to each hash table and the accelerator's performance speedup by using different number of entries.

4.3 Acceleration Optimization: Improving Memory Subsystem

In this section, we perform an analysis of the bottlenecks in the hardware accelerator for speech recognition presented in Section 4.2, and propose two architectural extensions to alleviate those bottlenecks. There are only two sources of pipeline stalls in the accelerator: misses in the caches and collisions in the hash tables. In case of a miss in the State, Arc or Token cache, the accelerator has to wait for main memory to serve the data, potentially introducing multiple stalls in the pipeline. On the other hand, resolving a collision in the hash table requires multiple cycles and introduces pipeline stalls, as subsequent tokens cannot access the hash until the collision is solved.

The results obtained by using our cycle-accurate simulator show that main memory latency has a much bigger impact on performance than the collisions in the hash tables. The performance of the accelerator improves by 2.11x when using perfect caches in our simulator, whereas an ideal hash with no collisions only improves performance by 2.8% over the baseline accelerator with the parameters shown in Table 4.1. Therefore, we focus our efforts on hiding the memory latency in an energy-efficient manner. In the following sub-section, we introduce an area-effective latency-tolerance technique that is inspired by decoupled access-execute architectures.

On the other hand, off-chip DRAM accesses are known to be particularly costly in terms of energy [39]. To further improve the energy-efficiency of our accelerator, we present a novel technique for saving memory bandwidth. This technique is able to remove a significant percentage of the memory requests for fetching states from the Fisher English WFST. Furthermore, we have seen similar saving by using other WFST datasets, such as Tedlium and Voxforge.

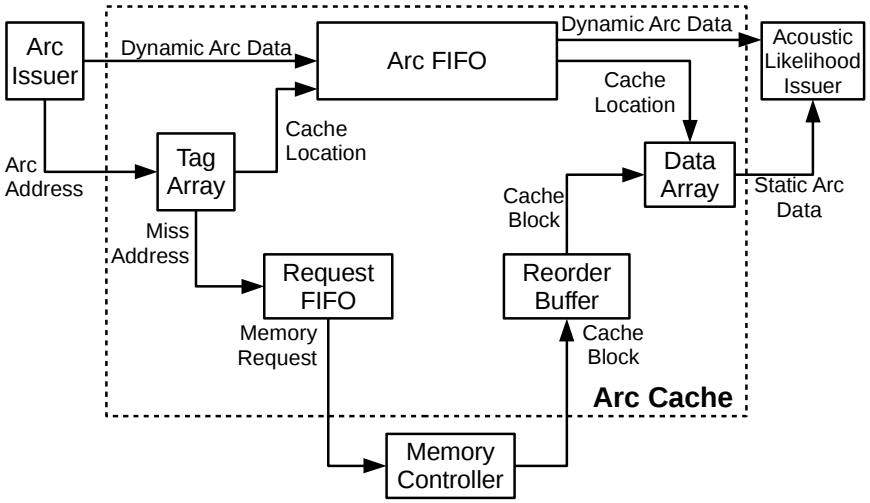


Figure 4.7: The data prefetching architecture for the Arc cache.

4.3.1 Data Prefetching: Hiding Memory Latency

Cache misses are the main source of stalls in the pipeline of the design and in consequence, main memory latency has a significant impact on the performance of the accelerator for speech recognition presented in Section 4.2. Regarding the importance of each individual cache, the results obtained in our simulator show that using a perfect Token cache provides a minor speedup of 1.02x. A perfect State cache improves performance by 1.09x. On the other hand, the Arc cache exhibits the biggest impact on performance, as removing all the misses in this cache would provide 1.95x speedup.

The impact of the Arc cache on the performance of the overall accelerator is due to two main reasons. First, the memory footprint for the arcs dataset is quite large: the Fisher English WFST has more than 34M arcs, whereas the number of states is around 13M. So multiple arcs are fetched for every state when traversing the WFST during the Viterbi search. Second, arc fetching exhibits poor spatial and temporal locality. Due to the pruning, only a small and unpredictable subset of the arcs are active on a given frame of speech. We observed that only around 25k of the arcs are accessed on average per frame, which represents 0.07% of the total arcs in the WFST. Hence, the accelerator has to fetch a different and sparsely distributed subset of the arcs on a frame basis, which results in large miss ratio for the Arc cache (see Figure 4.5). Therefore, efficiently fetching arcs from memory is a major concern for the accelerator.

The simplest approach to tolerate memory latency is to increase the size of the Arc cache, or to include a bigger second level cache. However, this approach causes a significant increase in area, power and latency to access the cache. Another solution to hide memory latency is hardware prefetching [69]. Nonetheless, we found that the miss address stream during the Viterbi search is highly unpredictable due to the pruning and, hence, conventional hardware prefetchers are ineffective. We implemented and evaluated different state-of-the-art hardware prefetchers [69, 32], and our results show that these schemes produce slowdowns and increase energy due to the useless prefetches that they generate.

Our proposed scheme to hide memory latency is based on the observation that arcs prefetching

can be based on computed rather than predicted addresses, following a scheme similar to the decoupled access-execute architectures [95]. After the pruning step, the addresses of all the arcs are deterministic. Once a state passes the pruning, the system can compute the addresses for all its outgoing arcs and prefetch their data from memory long before they are required, thus allowing cache miss latency to overlap with useful computations without causing stalls. Note that the addresses of the arcs that are required in a given frame only depend on the outcome of the pruning. Once the pruning is done, subsequent arcs can be prefetched while previously fetched arcs are being processed in the next pipeline stages.

Figure 4.7 shows our prefetching architecture for the Arc cache, which is inspired by the design of texture caches for GPUs [50]. The prefetching architecture processes arcs as follows. First, the Arc Issuer computes the address of the arc and sends a request to the Arc cache. The arc's address is looked up in the cache tags, and in case of a miss the tags are updated immediately and the arc's address is forwarded to the Request FIFO. The cache location associated with the arc is forwarded to the Arc FIFO, where it is stored with all the remaining data required to process the arc, such as the source token's likelihood. On every cycle, a new request for a missing cache block is sent from the Request FIFO to the Memory Controller, and a new entry is reserved in the Reorder Buffer to store the returning memory block. The Reorder Buffer prevents younger cache blocks from evicting older yet-to-be-used cache blocks, which could happen in the presence of an out-of-order memory system if the cache blocks are written immediately in the Data Array.

When an arc reaches the top of the Arc FIFO, it can access the Data array only if its corresponding cache block is available. Arcs that hit in the cache proceed immediately, but arcs that generated a miss must wait for its cache block to return from the memory into the Reorder Buffer. New cache blocks are committed to the cache only when its corresponding arc reaches the top of the Arc FIFO. At this point, the arcs that are removed from the head of the FIFO read their associated data from the cache and proceed to the next pipeline stage in the accelerator.

The proposed prefetching architecture solves the two main issues with the hardware prefetchers: accuracy and timeliness. The architecture achieves high accuracy because the prefetching is based on the computed rather than predicted addresses. Timeliness is achieved as cache blocks are not prefetched too early or too late. The Reorder Buffer guarantees that data is not written in the cache too early. Note that Arc Cache updates the tag array as soon as a miss is detected, and therefore this invalidates the cache line of the data array. Thus, the subsequent access to the same cache line whose data is already invalidated will raise a miss and a new request is inserted in the memory request FIFO. However, we have seen that this scenario is very rare as there is only 5% of increase in the number of accesses to the memory. Furthermore, if the number of entries in the Arc FIFO is big enough the data is prefetched with enough anticipation to hide memory latency. Based on our numbers, by setting the FIFO size to 64, we can cover almost all the memory latency, reaching the expected speedup as predicted for a prefect cache.

Our experimental results provided in Section 4.5 show that this prefetching architecture achieves performance very close to a perfect cache, with a negligible increase of 0.34% in the area of the Arc cache and 0.05% in the area of the overall accelerator.

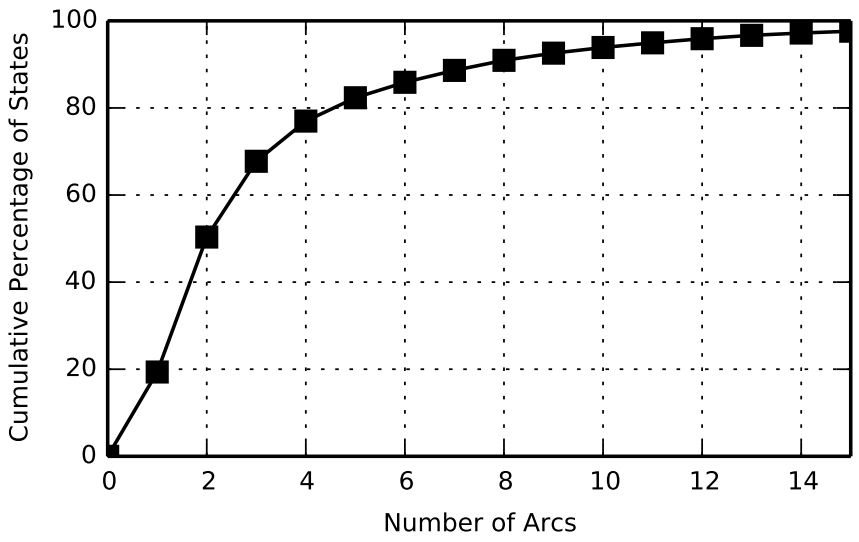


Figure 4.8: Cumulative percentage of states accessed dynamically vs the number of arcs. Although the maximum number of arcs per state is 770, 97% of the states fetched from memory have 15 or less arcs.

4.3.2 Memory Bandwidth Saving Technique

The accelerator presented in Section 4.2 consumes memory bandwidth to access states, arcs and tokens stored in off-chip system memory. The only purpose of the state fetches is to locate the outgoing arcs for a given state, since the information required for the decoding process is the arc's data. Each WFST includes an array of states that stores the index of the first arc and the number of arcs for each state. Accessing the arcs of a given state requires a previous memory read to fetch the state's data.

Note that this extra level of indirection is required since the states, for instance, in Fisher English WFST have different number of arcs ranging from 1 to 770. If all the states would have the same number of arcs, arcs indices could be directly computed from state index. Despite the wide range in the number of arcs, we have observed that most of the states accessed dynamically have a small number of outgoing arcs as illustrated in Figure 4.8. Based on this observation, we propose a new scheme that is based on sorting the states in the WFST by their number of arcs, which allows to directly compute arc addresses from the state index for most of the states.

Figure 4.9 shows the new memory layout for the WFST. We move the states with a number of arcs smaller than or equal to N to the beginning of the array, and we sort those states by their number of arcs. In this way, we can directly compute the arc addresses for states with N or less arcs. In the example of Figure 4.9, we use 4 as the value of N.

To implement this optimization, in addition to changing the WFST offline, modifications to the hardware of the *State Issuer* are required to exploit the new memory layout at runtime. First, N parallel comparators are included as shown in Figure 4.9, together with N 32-bit registers to store the values of S_1, $S_1 + S_2$... that are used as input for the comparisons with the state index. Second, a table with N entries is added to store the offset applied to convert the state index into its corresponding arc index in case the state has N or less arcs. In our example, a state with 2 arcs

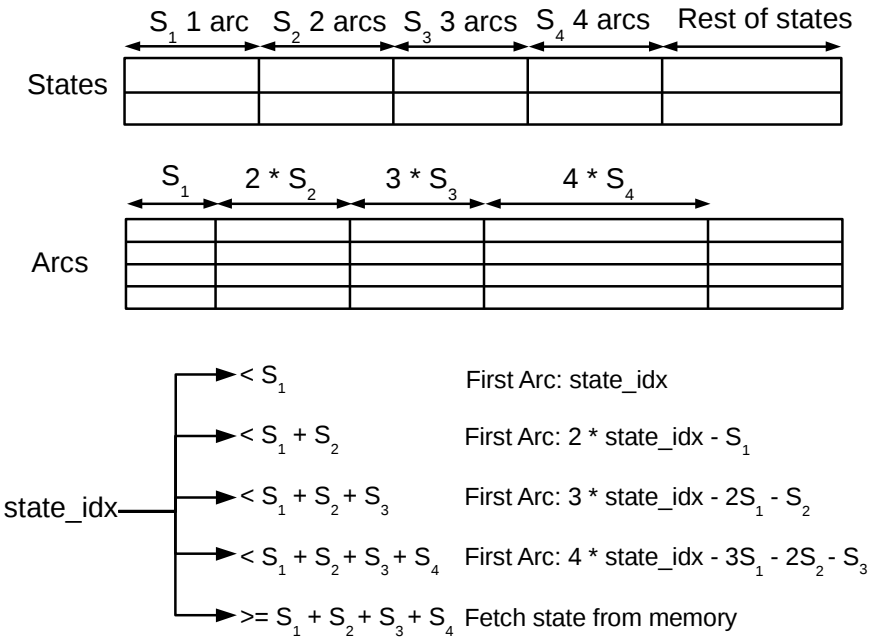


Figure 4.9: Changes to the WFST layout. In this example, we can directly compute arc index from state index for states with 4 or less arcs.

has an offset of $-S_1$, which is the value stored in the second entry of the table.

The outcome of the comparators indicates whether the arc index can be directly computed or, on the contrary, a memory fetch is required. In the first case, the result of the comparators also indicates the number of arcs for the state and, hence, it can be used to select the corresponding entry in the table to fetch the offset that will be used to compute the arc index. In addition to this offset, the translation from state index to arc index also requires a multiplication. The factor for this multiplication is equal to the number of arcs for the state, which is obtained from the outcome of the comparators. The multiplication and addition to convert state index into arc index can be performed in the Address Generation Unit already included in the State Issuer, so no additional hardware is required for those computations.

For our experiments we use 16 as the value of N. With this value we can directly access the arcs for more than 95% of the static states in Fisher English WFST and more than 97% of the dynamic states visited at runtime. This requires 16 parallel comparators, 16 32-bit registers to store the values of S_1 , $S_1 + S_2$ ... and a table with 16 32-bit entries to store the offsets. Our experimental results show that this extra hardware only increases the area of the State cache by 0.36% and the area of the overall accelerator by 0.02%, while it reduces memory bandwidth usage by 20%.

4.4 Power-Controlled Hash Architecture

In this section, we describe a novel power-control scheme applied to the accelerator's Hash tables. First, we analyze the potential of such technique for reducing the accelerator's power dissipation. As our evaluations show, the Hash tables dissipate nearly 53% of the total power,

including both the leakage and dynamic activities. Thus, we focus on controlling the power of these modules as they represent the main bottleneck. On the other hand, the static power accounts for 72% of the accelerator's total power and almost 50% of this amount is dissipated by the Hash tables. Therefore, we target decreasing the Hash tables' leakage power, in order to efficiently lower both the accelerator's power and energy consumption.

Regarding static power optimization, power gating [77] is a very effective technique used in many designs. However, we cannot apply this approach for the Hash tables since it would loose the data, which is needed for the next frame's evaluation. The reason is that these tables use a well-dispersed hashing function, which therefore distributes all the tokens almost uniformly across the table entries. Regarding the backup buffers, due to the consecutive allocation of their entries, power-gating can be only used for a moderate portion on average.

Another less aggressive method of reducing the leakage energy consumption is to put buffers into a low-power drowsy mode in which the state of memory is preserved. To do so, two schemes have been introduced at the circuit level: one using adaptive body-biasing with multi-threshold CMOS [70]; and the other one chooses between two different voltages for each operation mode [30]: drowsy voltage and active voltage. Because of the short-channel effects in deep-submicron process and the low operating voltage (for the second approach), a significant reduction of the leakage power is yielded, achieving most of the advantage of a power-gating strategy while preserving the data.

As aforementioned, we can use power-gating for the backup buffer, partially, by conservatively selecting the partitions to turn off. A more aggressive approach would incur in some performance degradation in case the active partitions do not suffice to store all active states, and some partitions need to be reactivated. Consequently, we exploit the drowsy technique as the base of our power-control scheme for both Hash tables and backup buffers. We based our implementation on the approach which uses dual voltages as it is simpler and more effective.

Regarding the switching between drowsy and active modes for the entire Hash structure, we use a simple yet effective policy which is based on the access pattern of the memory partitions. Unlike the static scheme proposed in [30], we use a dynamic threshold for transitioning a partition from active to drowsy mode. Moreover, we explore the optimal number of partitions for both the Hash table and backup buffer. The power of each partition is controlled by its own drowsy control logic.

Our power-control system works as follows. Whenever there is an access to a partition of the hash table, it is switched on by setting the voltage of that partition to the active value (1 V for 28 nm technology). Then, a counter, called change-mode counter, is initiated in order to measure a statically-adjusted period. The counter starts over at each access to the same partition. As the resetting of the counter happens repetitively because of consecutive accesses to the same partition, the active period varies dynamically throughout the execution time and for the different partitions. When the change-mode counter exceeds a given threshold, the partition's voltage is set to the drowsy mode, lowering the voltage to 0.3V, for which the state of all the entries is preserved. To manage the power-control mechanism of each partition, a very simple logic is required in addition to the change-mode counter: two pass transistors for connecting different voltages to the partition, and a 1-bit SRAM cell to keep track of the drowsy control signal.

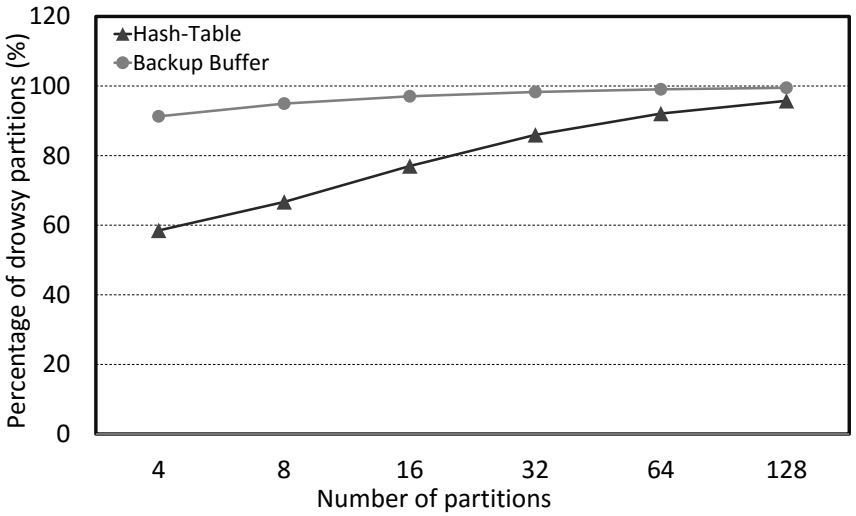
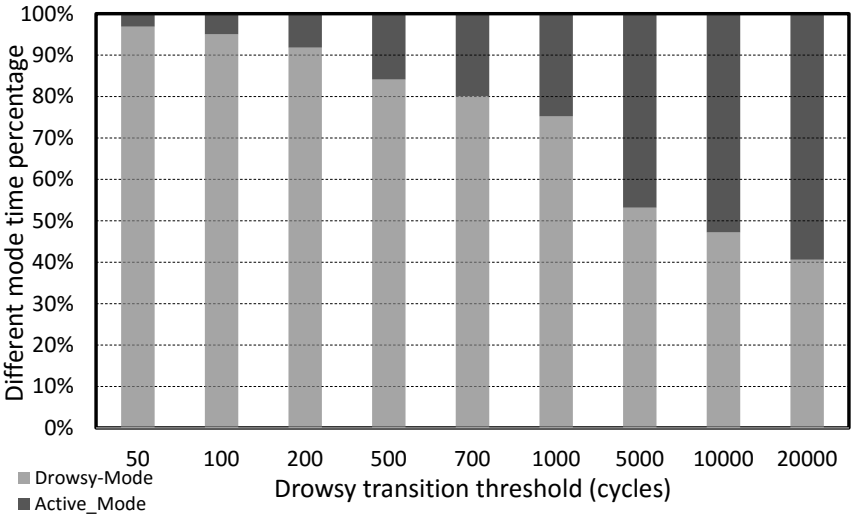


Figure 4.10: Percentage of hash in drowsy mode for different number of partitions.

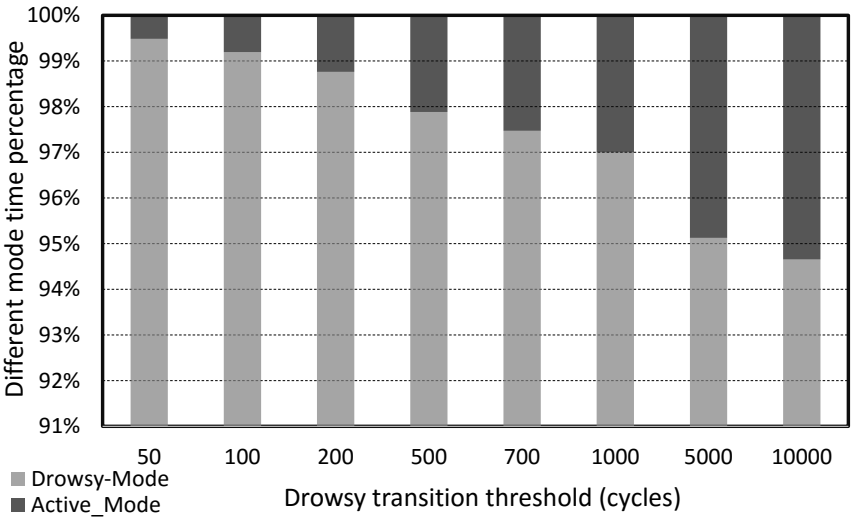
Figure 4.10 shows the average percentage of partitions that are in drowsy mode when decoding speech signal for different numbers of partitions in the hash table and backup buffer. As it can be seen, the more partitions we use for the Hash table, the larger the percentage that stays in the drowsy mode. The main reason is that the hash function maps tokens to entries in a nearly uniformly-distributed manner. Consequently, as we increase the number of partitions, i.e. decreasing the size of each partition, they are less frequently accessed, which results in a higher drowsy percentage. On the other hand, accesses to the backup buffer are more predictable since partitions are written in order starting from the first entry. Thus, although the buffer can be in the drowsy mode more by increasing the number of partitions, the improvement is not as huge as for the hash table.

In order to further evaluate our technique, we have explored various drowsy mode transitioning thresholds to decide when an active partition is switched to drowsy mode. Figure 4.11 shows the breakdown of the percentage of time in which a partition is in drowsy or active mode, on average, for different transition thresholds. As it can be seen, by increasing the threshold, which determines the minimum cycles that a partition is active, the percentage of drowsy mode decreases in either Hash table or backup buffer. This reduction is substantially larger for the Hash table because of the more frequent accesses to each partition when using a longer period, which triggers more resets of the change-mode counter, resulting in a higher percentage for the active mode operation. On the other hand, by setting an aggressive period such as 50 cycles for both the hash table and backup buffer, it imposes negligible overhead, as the delay and energy cost of switching a partition to different operation modes are relatively small. Based on our H-SPICE simulation results, each transition takes 0.12 ns (much less than a cycle) and consumes 2.83×10^{-15} J energy at 28 nm technology. Furthermore, we can hide almost all the delay of turning on a partition by notifying the hash table in advance when a request for either updating or adding a token is received in the Token Issuer.

By using the aforementioned exploration and statistics, we decided to use 128 and 32 partitions in the Hash table and backup buffer, which results in 95.8% and 98.3% of drowsy mode operation on



(a) Hash table



(b) Backup buffer

Figure 4.11: Percentage of drowsy vs active time for several transition thresholds considering 128 partitions in each buffer.

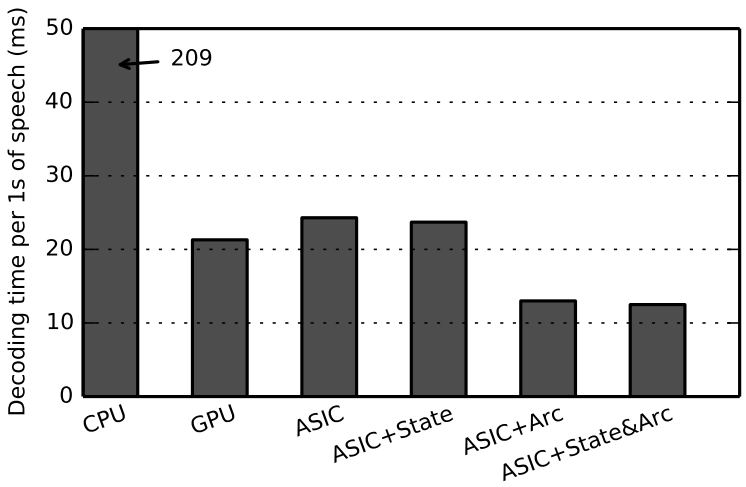


Figure 4.12: Decoding time, i. e. time to execute the Viterbi search, per second of speech. Decoding time is smaller than one second for all the configurations, so all the systems achieve real-time speech recognition.

average, respectively. Moreover, we set the transition threshold as 50 cycles (83 ns) due to its good trade-off between saving leakage power and energy overhead of switching between different operation modes. Furthermore, we extended the Hash's microarchitecture in a way to add a snooping port that the Token Issuer uses to report a future access to a drowsy partition. By notifying the access to the partition in advance in the pipeline, our accelerator is able to hide the transition delay. Regarding the implementation of this technique, the area of the Hash table is increased by 0.16% while only incurring in a 0.05% overhead in the total accelerator's area.

4.5 Experimental Results and Comparison

In this section, we provide details on the performance and energy consumption of the CPU, the GPU and the different versions of the accelerator. At first, we only consider Viterbi search and report our evaluation for six different configurations. *CPU* corresponds to the software implementation running on the CPU described in Table 3.1. *GPU* refers to the CUDA version running on the high-end GPU presented in Table 3.2. *ASIC* is our accelerator described in Section 4.2 with parameters shown in Table 4.1. *ASIC+State* corresponds to the bandwidth saving technique for the State Issuer presented in Section 4.3.2. *ASIC+Arc* includes the prefetching architecture for the Arc cache presented in Section 4.3.1. Finally, the configuration labeled as *ASIC+State&Arc* includes our techniques for improving both the State Issuer and the Arc cache. Furthermore, we consider the whole ASR benchmark and compare our numbers with the mobile GPU configured with the parameters of Table 3.3.

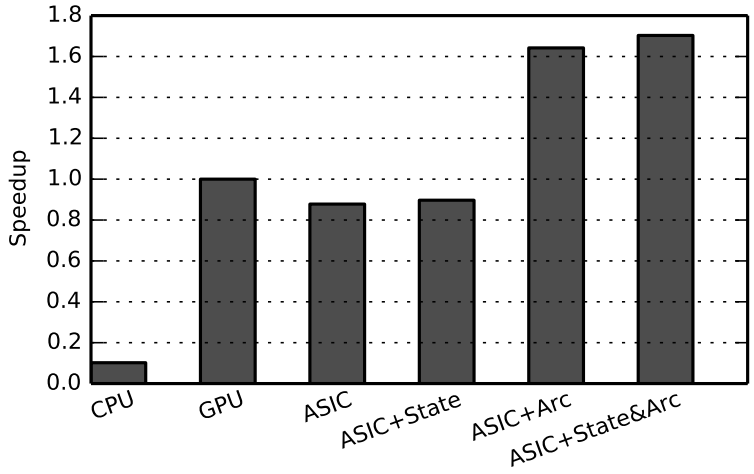


Figure 4.13: Speedups achieved by the different versions of the accelerator. The baseline is the GPU.

4.5.1 Viterbi Acceleration Comparison With Desktop CPU and GPU

Figure 4.12 shows the decoding time per one second of speech, a common metric used in speech recognition that indicates how much time it takes for the system to convert the speech waveform into words per each second of speech. As illustrated, all the systems achieve real-time speech recognition, as the processing time per one second of speech is significantly smaller than one second. Both the *GPU* and the *ASIC* provide important reductions in execution time with respect to the *CPU*. The *GPU* improves performance by processing multiple arcs in parallel. The *ASIC* processes arcs sequentially, but it includes hardware specifically designed to accelerate the search process, avoiding the overheads of software implementations.

Figure 4.13 shows the speedups with respect to the *GPU* for the same configurations. The initial design of the *ASIC* achieves 88% of the performance of the *GPU*. The *ASIC+State* achieves 90% of the *GPU* performance. This configuration includes a bandwidth saving technique that is very effective for removing off-chip memory accesses as reported later in this section, but it has a minor impact on performance (its main benefit is power reduction as we will see later). Since our accelerator processes arcs sequentially, performance is mainly affected by memory latency and not memory bandwidth. On the other hand, the configurations using the prefetching architecture for the Arc cache achieve significant speedups, outperforming the *GPU*. We obtain 1.64x and 1.7x speedup for the *ASIC+Arc* and *ASIC+State&Arc* configurations respectively over the GPU (about 2x with respect to the ASIC without these optimizations). The performance benefits come from removing the pipeline stalls due to misses in the Arc cache, as the data for the arcs is prefetched from memory long before they are required to hide memory latency. The prefetching architecture is a highly effective mechanism to tolerate memory latency, since it achieves 97% of the performance of a perfect cache according to our simulations.

Our accelerator for speech recognition provides a huge reduction in energy consumption as illustrated in Figure 4.14. The numbers include both static and dynamic energy. The base *ASIC* configuration reduces energy consumption by 171x with respect to the *GPU*, whereas the optimized version using the proposed improvements for the memory subsystem (*ASIC+Arc&State*)

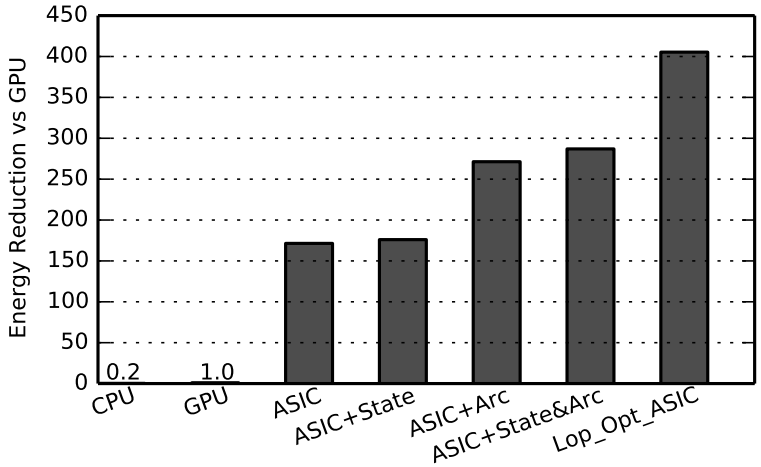


Figure 4.14: Energy reduction vs the GPU, for different versions of the accelerator.

reduces energy by 287x. The reduction in energy comes from two sources. First, the accelerator includes dedicated hardware specifically designed for speech recognition and, hence, it achieves higher energy-efficiency for that task than general purpose processors and GPUs. Second, the speedups achieved by using the prefetching architecture provide a reduction in static energy. The last configuration, named as *Lop_Opt_ASIC*, includes the Hash's power-control mechanism in addition to all the previous optimizations. The *Lop_Opt_ASIC* reduces the energy of the accelerator by 118x with respect the GPU configuration. This additional decrease in energy is due to controlling the leakage power of the Hash tables by switching almost 97% of its partitions to the drowsy mode (see Figure 4.11).

On the other hand, Figure 4.15 shows the average power dissipation for the different systems, including both static and dynamic power. The *CPU* and the *GPU* dissipate 32.2 W and 76.4 W respectively when running the speech recognition software. Our accelerator provides a huge reduction in power with respect to the general purpose processors and GPUs, as its power dissipation is between 321 mW and 462 mW depending on the configuration. The prefetching architecture for the Arc cache increases power dissipation due to the significant reduction in execution time that it provides, as shown in Figure 4.13. With respect to the initial *ASIC*, the configurations *ASIC+Arc* and *ASIC+State&Arc* achieve 1.87x and 1.94x speedup respectively. The hardware required to implement these improvements to the memory subsystem dissipates a very small percentage of total power. The Arc FIFO, the Request FIFO and the Reorder Buffer required for the prefetching architecture dissipate 4.83 mW, only 1.07% of the power of the overall accelerator. On the other hand, the extra hardware required for the State Issuer (comparators and table of offsets) dissipates 0.15 mW, 0.03% of the total power. In the best case, *Lop_Opt_ASIC* configuration achieves 29.2% power reduction versus the optimized version of accelerator, by only dissipating 5.3 mW (1.65% of the total power) for the change-mode counters and the drowsy control logic.

The memory bandwidth saving technique presented in Section 4.3.2 avoids 20% of the accesses to off-chip system memory as illustrated in Figure 4.17. The baseline configuration for this graph is the initial *ASIC* design. The figure also shows the traffic breakdown for the different types of data stored in main memory: states, arcs, tokens and the overflow buffer. Our technique targets the

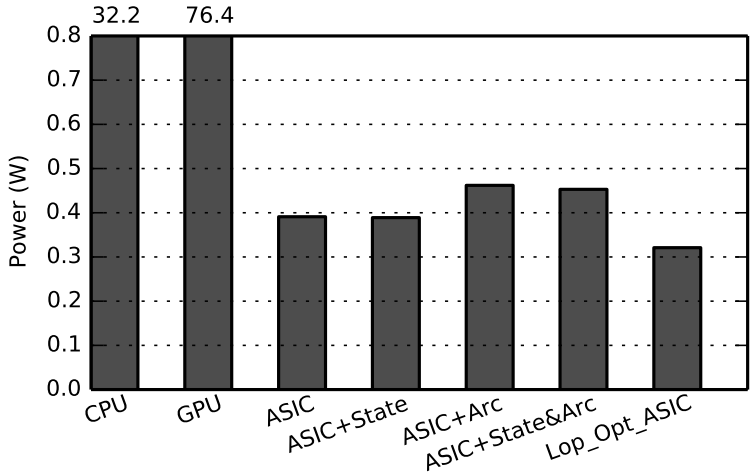


Figure 4.15: Power dissipation for the CPU, GPU and different versions of the accelerator.

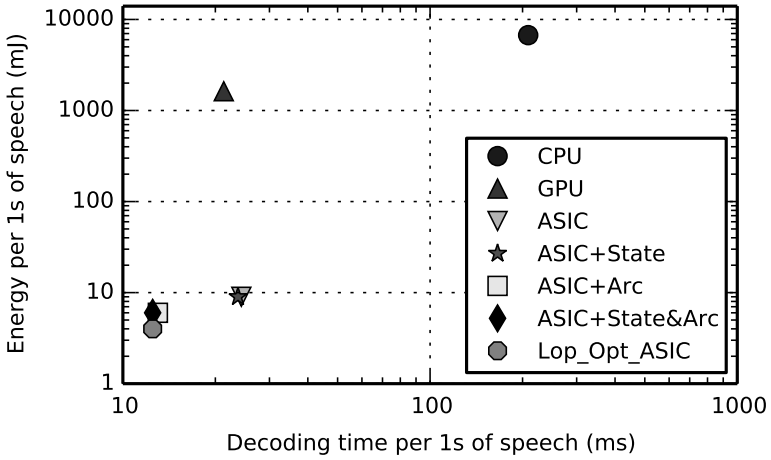


Figure 4.16: Energy vs decoding time per one second of speech. The measurements are collected based on Kaldi's Fisher English dataset.

memory accesses for fetching states, which represent 23% of the total traffic to off-chip memory. As it can be seen in the figure, our technique removes most of the off-chip memory fetches for accessing the states. The additional hardware included in the State Issuer is extremely effective to directly compute arc indices from state indices for the states with 16 or less arcs, without issuing memory requests to read the arc indices from main memory for those states. Note that in Figure 4.17 we do not include the configurations that employ the prefetching architecture, as this technique does not affect the memory traffic. Our prefetcher does not generate useless prefetch requests as it is based on computed addresses.

To sum up the energy-performance analysis, Figure 4.16 plots energy vs execution time per one second of speech. As it can be seen, the *CPU* exhibits the highest execution time and energy consumption. The *GPU* improves performance in one order of magnitude (9.8x speedup) with respect to the *CPU*, while reducing energy by 4.2x. The different versions of the accelerator

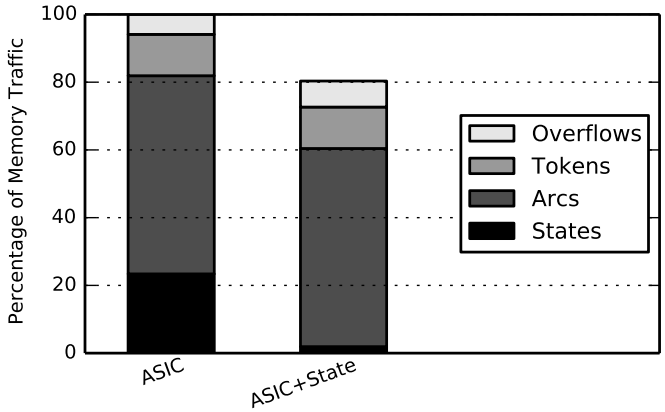


Figure 4.17: Memory traffic for the baseline ASIC and the version using the optimization for the state fetching presented in Section 4.3.2.

achieve performance comparable or higher to the *GPU*, while providing an energy reduction of two orders of magnitude. Regarding the effect of the techniques to improve the memory subsystem, the prefetching architecture for the Arc cache provides significant benefits in performance and energy. On the other hand, the aim of the memory bandwidth saving technique for the State Issuer is to reduce the number of accesses to off-chip system memory. This technique achieves a reduction of 20% in the total number of accesses to off-chip DRAM as reported in Figure 4.17. When using both techniques (configuration labeled as *ASIC+State&Arc*) the accelerator achieves 16.7x speedup and 1185x energy reduction with respect to the *CPU*. Compared to the *GPU*, this configuration provides 1.7x speedup and 287x energy reduction. Finally, the configuration with the low-power technique keeps the same performance as the ASIC with all the other optimizations, while deceasing the energy by another two orders of magnitude, 1674x and 405x according to *CPU* and *GPU* respectively.

Finally, we evaluate the area of our accelerator for speech recognition. The total area for the initial design is 24.07 mm^2, a reduction of 16.53x with respect to the area of the NVIDIA GeForce GTX 980 (the die size for the GTX 980 is 398 mm^2 [33]). The hardware for the prefetching architecture in the Arc cache, i. e. the two FIFOs and the Reorder Buffer, produce a negligible increase of 0.05% in the area of the overall accelerator. The extra hardware for the State Issuer required for our bandwidth saving technique increase overall area by 0.02%. The power-control approach of the hash table represents only 0.04% of area overhead. In total, the area for the accelerator including all the optimizations is 24.11 mm^2.

4.5.2 Accelerated ASR Evaluation With Respect to a Mobile GPU

Figure 4.18 shows execution time and energy consumption for the overall ASR system, including the DNN and the Viterbi search. The GPU-only configuration corresponds to a system that runs the entire ASR pipeline on the mobile GPU (top of Figure 4.2). The GPU+ACC configuration is the system that employs the GPU for DNN computation and our accelerator for the Viterbi search, as illustrated in Figure 4.4. The GPU+ACC configuration provides 5.26x speedup over

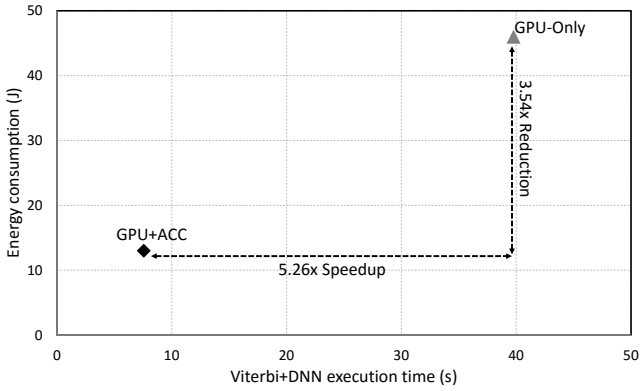


Figure 4.18: Decoding time versus energy consumption of the whole ASR on the GPU and our system.

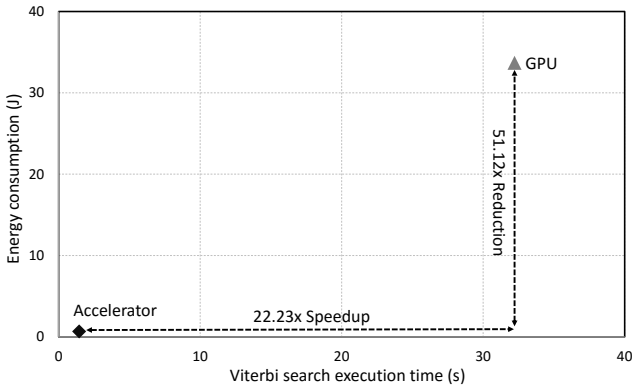


Figure 4.19: Decoding time versus energy consumption running Viterbi search on the GPU and accelerator.

the GPU-only system, which comes from two sources. Firstly, the accelerator provides a large speedup for the main bottleneck of ASR, the Viterbi search. Figure 4.19 shows the decoding time and energy of the Viterbi search running on the accelerator and GPU. As seen, Viterbi accelerator achieves real-time performance by a wide margin, as it takes 11 ms to decode each second of speech. The speedup achieved by the accelerator over the mobile GPU is 22x. In addition to the performance improvement, the accelerator also provides a huge reduction (48x) in energy consumption. Secondly, the GPU+ACC configuration overlaps the execution time of the DNN and the Viterbi search, as illustrated in Figure 4.2, providing further performance improvements over the GPU-only configuration that has to serialize the two stages. On the other hand, GPU+Acc provides an energy reduction of 3.54x.

5

UNFOLD: Memory-Efficient ASR System

In the previous chapter, we propose an accelerator design which provides high-performance architecture for ASR, consumes very low energy and requires a small power budget. However, it suffers from an important issue that severely constrains its applicability for mobile and IoT devices: the huge size of the WFST. Throughout this chapter, we go through a different approach for Viterbi acceleration, called as UNFOLD, which reduces the memory usage by an order of magnitude (31x). As it is mentioned in Section 1.2.2, the WFST takes more than 87% of the memory footprint of the ASR system. In order to reduce the vast requirement of WFST storage, we apply on-the-fly composition of the two decomposed WFST graphs, called as Acoustic-Model (AM) and Language-Model (LM). Moreover, we apply some simple and efficient compression techniques in order to further shrink the size of speech datasets.

As aforementioned in Chapter 2, on-the-fly composition is an alternative to the fully-composed approach used for the Viterbi acceleration design of the previous chapter. In spite of having the huge memory requirement for storing the vast fully-composed WFST, on-the-fly composition scheme has the main benefit of reducing the memory footprint significantly, as it uses the decomposed sub-WFSTs for constructing the Viterbi search. These graphs take around 100 MB of memory space on average considering different ASR decoders, whereas the enormous size in the order of Gigabytes for the fully-composed WFST. Moreover, as the size of the speech dataset decreases by a great deal, we save a lot of memory accesses, which translates into significant power reduction (2x) and also energy-saving (28%). On the other hand, this will introduce some computational complexity for the ASR decoder, since several WFSTs (two for UNFOLD's implementation) are expanded during the search process in order to compose the knowledge resources where ever necessary. The Viterbi algorithm explores the AM WFST in the same way as the fully-composed WFST. However, in case of reaching an AM's arc associated with a word ID, AM and LM get composed on-the-fly, i.e. the LM WFST is searched from the previous history state in order to update the new word history.

UNFOLD is designed in a way to efficiently handle the composition of the AM and LM WFSTs. In Section 5.1, we describe the main modifications made to the Viterbi accelerator in order to support on-the-fly composition operations. Next, we define our compression techniques in Section 5.2, to reduce the size of the speech dataset. Moreover, we explain the logic added at the different accelerator's components in order to uncompress the WFST data at the minimum performance and energy cost.

In order to further reduce UNFOLD's memory bandwidth requirements, we introduce an orthogonal extension to the Viterbi expansion by considering the frame-to-frame locality and the search-confidence, called as Locality AWare Scheme (LAWS). Initially, we propose to manage the ASR's workload by taking into account the search's hypotheses data-locality besides using the beam distance of the Viterbi beam search, reaching a sub-optimal trade-off between accuracy and energy-consumption. Secondly, we try to improve this trade-off by the help of receiving the search-confidence feedback during the processing of Viterbi algorithm. We define the confidence term according to the number of hypotheses evaluated per frame: When the number hypotheses is high, we say that the ASR system has low confidence, since there are many alternative hypotheses to explore, whereas we say that the confidence is high when this number is low. In the end, we develop an adaptive approach to dynamically adapt the amount of Viterbi's workload based on the search-confidence and data-locality at each frame.

We have measured nearly 86% of data-locality between consecutive frames on average, by evaluating the entire 5.4 hours of speech of the Librispeech corpus. Moreover, we have seen that the "correct" hypothesis at each frame, i.e. the hypothesis that ends up being part of the final answer which is not known until the end of processing each utterance, often resides in the on-chip memory. Note that the best hypothesis at each frame can be different from the correct one, since the correct one may have lower likelihood at some frames while still being in the most likely full-hypothesis from the beginning to the end of recognizing an utterance. Furthermore, we have observed that the correct hypothesis and the best one at each frame are often the same, and even when they differ, their scores (i.e. likelihoods) are quite close.

This behaviour of ASR systems has motivated us to propose a scheme that uses a more restrictive beam when the explored hypotheses require off-chip memory accesses. The rationale behind it is that these accesses are very expensive and based on the above observations are very unlikely to contain the correct hypotheses. Locality-AWare Scheme (LAWS) dynamically adjusts the beam distance for ASR based on the hypotheses' data-locality. In other words, our scheme uses the locality and hypotheses' likelihood to decide the search strategy, unlike previous schemes that use only likelihood information. This results in a more efficient expansion on the search graph in terms of accuracy and energy consumption.

LAWS uses a dynamic beam adaptation policy, as stated above. This policy not only takes into account the locality of the data, besides the likelihood scores, but also the confidence of the ASR system at each frame. We go through the full details of the LAWS mechanism in Section 5.3.

We finally conclude this chapter in Section 5.4, by showing some experimental results and analyzing the benefits of the UNFOLD's acceleration design and also the LAWS extention to the UNFOLD's architecture.

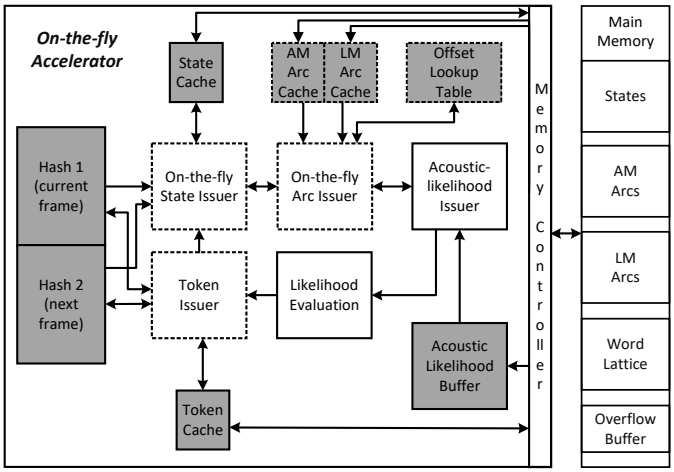


Figure 5.1: The architecture of the UNFOLD's design. Memory components are in gray color and the modified components with respect to [109] are shown with dashed lines.

5.1 UNFOLD's Architecture

Figure 5.1 shows an overview of the UNFOLD's design, which is based on the previous approach presented for the fully-composed WFSTs (see Chapter 4). The accelerator includes several pipeline stages to fetch states, i.e. On-the-fly State Issuer, fetch arcs, i.e. On-the-fly Arc Issuer, fetch DNN/GMM acoustic scores, i.e. Acoustic-likelihood Issuer, compute the likelihood of a hypothesis, i.e. Likelihood Evaluation, and write the data of the new tokens, i.e. Token Issuer. Furthermore, UNFOLD provides several on-chip caches for the different datasets used in ASR, an *Offset Lookup Table* to speed up finding the location of the LM arcs, and two Hash Tables that store the tokens for the current and the next frames of speech. All the memory components are in gray color and the newly modified stages with respect to the Viterbi accelerator of Chapter 4 are enclosed with dashed lines.

Compared with our first design presented in Chapter 4, we have modified the State and Arc Issuers to fetch data from two WFSTs instead of one fully-composed WFST, as UNFOLD operates on the individual AM and LM models. We read the states and arcs from the AM as in the fully-composed WFST and, therefore, it requires no change with respect to the previous version. However, fetching an LM's arc is more complex, since the arc associated with a specific word ID has to be found among the thousands of outgoing arcs of a given state. We have seen that by using a simple linear search, the Viterbi search experiences almost 10x performance drop comparing to the fully-composed acceleration approach of Chapter 4. In order to perform this search more efficiently, we store the outgoing arcs of each state sorted by their word ID in main memory, and the Arc Issuer is extended to implement a binary search.

To further reduce the overhead of locating arcs in the LM, we observe that accesses to the LM exhibit good temporal locality, i.e. if a word is recognized in a given hypothesis, it is likely to be recognized in other hypotheses in the near future. We exploit this temporal locality with a new table, called *Offset Lookup Table*, that stores the arcs' offsets associated with the recently used combinations of LM state and word ID, i.e. it stores the results of recent binary searches to be

Table 5.1: Accelerator's configuration parameters.

Technology	28/32 nm
Frequency	800 MHz
State Cache	256 KB, 4-way, 64 B/line
AM Cache	512 KB, 8-way, 64 B/line
LM Cache	32 KB, 4-way, 64 B/line
Token Cache	128 KB, 2-way, 64 B/line
Acoustic Likelihood Buffer	64 Kbytes
Hash Table	576 KB, 32K entries
Offset Table	192KB, 32K entries
Memory Controller	32 in-flight requests
Likelihood Evaluation Unit	4 floaing-point adders, 2 floaing-point comparators

reused by subsequent arc fetches. The accelerator first probes this table when looking for an LM arc and in case of a hit, the offset of the arc is obtained from the table in one cycle and no binary search is required. We show later in the paper that a small *Offset Lookup Table* achieves high hit ratios, avoiding the overhead of locating arcs in the LM by a large extent.

Another modification introduced in our design is the use of two independent caches for fetching arcs from AM and LM models in parallel, avoiding stalls in the pipeline of the accelerator. Due to the aforementioned complexity for fetching LM arcs, we found it beneficial for performance to have a dedicated cache for this task. Note that a single Arc Cache with two read ports could have been used as an alternative, but it does not provide any advantage as the AM and LM datasets are disjoint, i.e. there is no data replication in our separated arc caches. Regarding the states, we share a single cache for both AM and LM due to several reasons. First, the states' dataset is significantly smaller than the arc's dataset (less than 12% of the WFST size). Second, the pressure on the State cache is low, as concurrent requests for AM and LM states are uncommon. We have observed that splitting the State cache into two separate caches for AM and LM does not provide any noticeable performance or energy improvement.

On the other hand, we also reduce the size of the different caches of the accelerator without any increase in miss ratio, since the new dataset is considerably smaller due to the use of on-the-fly composition. Finally, we modify the Token Issuer to write the word lattice in a compact representation as described in [80], which provides further savings in memory traffic. Regarding the ASIC parameters, we have conducted a full design space exploration in order to set the sizes of the different on-chip memory components. Table 5.1 shows the detailed configuration of the UNFOLD's architecture. As seen, the sizes of the buffers and caches are considerably reduced compared with the previous Viterbi accelerator design, which as we show later will result in much less power dissipation for the whole accelerator.

5.1.1 UNFOLD's Pipeline

We use the example shown in Figure 2.10 to illustrate how UNFOLD performs the Viterbi search while composing the AM and LM models on-the-fly. The pipeline starts from the hash table

containing the source tokens for the current frame of speech (i.e., all the states expanded by the previous frame). Assuming we are at frame 2, this hash table includes the information of four tokens: (2, 1), (5, 2), (0, 2) and (7, 0). These tokens represent four different hypotheses, i.e. four alternative representations of the speech up to the current frame. The State Issuer reads the tokens from the hash table and applies a pruning step, discarding tokens whose likelihood is smaller than a threshold. This threshold is updated on each frame according to the best-hypothesis's probability and a predefined beam value. In this example, the four tokens pass the pruning and, thus, the State Issuer fetches the information of states 2, 5, 0 and 7 from the AM WFST by using the State Cache.

On the next stage, the Arc Issuer fetches the outgoing arcs of each AM state through the AM Arc Cache. Arcs departing from states 2, 5 and 0 have epsilon word ID (see Figure 2.10) and, hence, no information from LM is required. However, the outgoing arc of state 7 is associated with the word "*THREE*". In this case, token (7, 0) has to transition in both models by traversing the corresponding arcs in AM and LM. Therefore, a look-up in the LM is required to fetch the outgoing arc of state 0 associated with the word "*THREE*". To locate this arc, the Arc Issuer first looks for the (LM state, word) combination of (0, "*THREE*") at the Offset Lookup Table. In case of a hit, the arc offset associated with that state and word ID is retrieved from the table and the address of the arc in main memory is obtained. Otherwise, the Arc Issuer starts a binary search to find the outgoing arc of state 0 whose input label equals word "*THREE*". Another independent cache, called LM Arc Cache, is utilized to speed up the search. Note that this process may take several cycles, as multiple memory fetches may be necessary to find the arc. Once the arc is located, its offset is stored in the *Offset Lookup Table* to be later reused by subsequent fetches. The weight of this LM arc represents the unigram likelihood of the word "*THREE*", and it is used to rescore the likelihood of the new token (8, 3), expanded from (7, 0).

The rest of the pipeline basically follows the behavior of the fully-composed accelerator [110], as described in Chapter 4. The Acoustic Likelihood Issuer gets the acoustic scores of the phonemes associated with the AM arcs. These scores are computed by the DNN/GMM on the GPU and stored in the Acoustic Likelihood Buffer. Next, the Likelihood Evaluation unit performs the required processing to compute the probability of the new hypothesis, combining the different likelihoods from the source token, AM arc and LM arc. Finally, the Token Issuer stores the new token's information. The information which is only required for the next frame is stored in the hash table, whereas the information required for generating the word lattice is written in main memory by using the Token Cache. The hash tables are indexed through a combination of IDs of AM and LM states associated with the token. Other mechanisms of the hash structure, like handling collisions and overflows, work as described in Section 4.2.1.

5.1.2 Optimizing Back-off Processing for the LM Expansion

As mentioned in Section 2.3.2, not all possible combinations of two and three words are included in the LM, as it would require a huge amount of memory. Combinations that are very unlikely are omitted to keep the LM manageable. Therefore, there are cases where the Arc Issuer cannot find an arc associated with a particular word ID from the outgoing arcs of a state in LM. For these cases, the LM provides a back-off arc in each state. These arcs back off from trigram to bigram, or

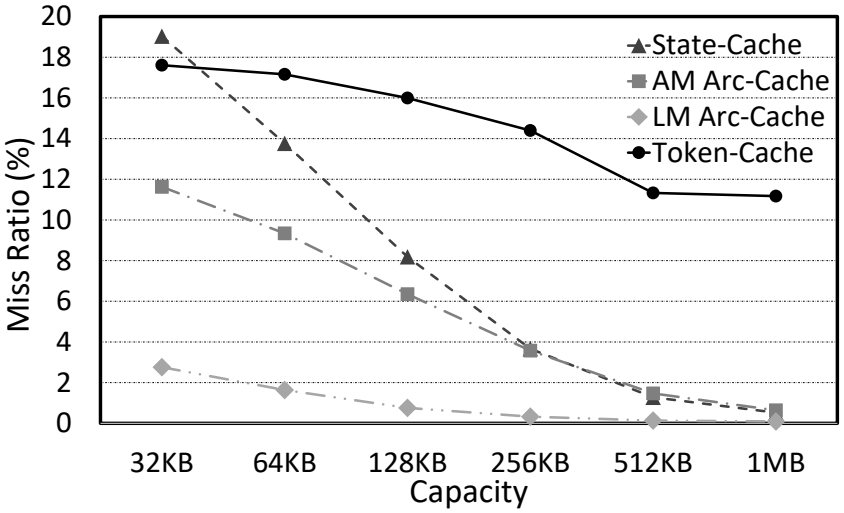


Figure 5.2: Miss-ratio vs capacity of the several caches

from bigram to unigram states. All the unigram likelihoods are maintained to guarantee that, in the worst case, any word ID can be found in an arc departing from state 0. For instance, consider the word sequence "*TWO-ONE*" in Figure 2.10 (being at state 5), if the next word is "*TWO*", then we use a back-off transition to state 1 which represents the unigram history of seeing word "*ONE*" since there is no 3-gram model for "*TWO-ONE-TWO*" from state 5. Next, as there is no bigram from state 1 for the word "*TWO*", another back-off transition is taken to state 0. Then, by traversing the right arc, it reaches to the destination state 2, which corresponds to having seen the unigram "*TWO*".

The back-off mechanism increases the workload in UNFOLD, as it requires multiple arc-fetches in different states for a single token, with potentially multiple binary searches if misses arise in the *Offset Lookup Table*. To optimize the back-off mechanism, we introduce a preemptive pruning scheme in the Arc Issuer, which prunes in advance the backed-off hypotheses whose probabilities go lower than a threshold. Back-off arcs include a weight, i.e. likelihood, to penalize the hypotheses, as they are traversed for very unlikely combinations of words that were omitted from the LM. For this purpose, the Arc Issuer updates and checks the likelihood of a hypothesis after traversing a back-off arc, in order to stop the back-off mechanism as soon as the hypothesis can be safely discarded. Since the likelihood of a hypothesis is a monotonically decreasing function, it is guaranteed that we only discard the hypotheses that would be pruned away later in the accelerator.

Our preemptive pruning requires modest hardware, as only three floating-point units are added in the Arc Issuer to apply the weight of the AM arc, the back-off arc and compare the result with the threshold for the pruning. Our experimental results show that, on average, 22.5% of the total number of hypotheses are pruned away and 16.3% performance improvement is achieved with the preemptive pruning.

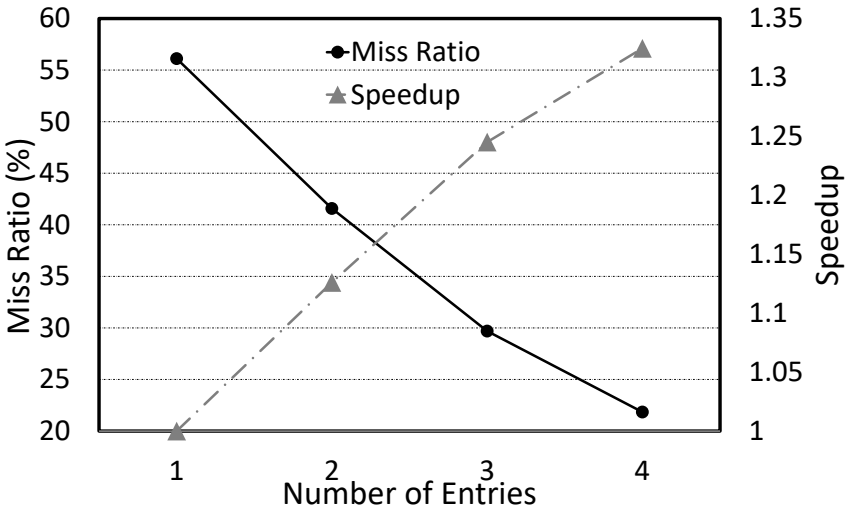


Figure 5.3: The effect of Offset Lookup Table's size on speedup

5.1.3 Analysis of the Four Caches and Offset Lookup Table

Figure 5.2 shows the miss ratio versus capacity for the different caches included in the accelerator. Regarding the State Cache and the two Arc Caches, AM and LM, the miss ratio is significantly reduced when increasing capacity, reaching a miss ratio smaller than one percent for the size of one Megabyte. For the Token Cache, miss ratio is close to 12% even for a capacity of one Megabyte due to compulsory misses. New tokens are written in consecutive memory locations and, thus, spatial locality is exploited at the cache line level. However, once a line is filled with tokens it is unlikely to be reused, i.e. accesses to the tokens exhibit poor temporal locality.

The miss ratios shown in Figure 5.2 are smaller than the ones reported in Figure 4.5 for the fully-composed WFST, since the memory footprint has been largely reduced in our proposal through on-the-fly composition and compression (see Section 5.2). For a State Cache of 512 KB, the miss ratio is reduced from 29% for the previous accelerator to 1.5% in UNFOLD. On the other hand, an Arc Cache of 1 MB exhibits a 27% miss ratio for the fully-composed WFST, whereas the two separated AM and LM arc caches of 512 KB achieve 1.6% and 0.3% miss ratios respectively. Note that the state and arc datasets are reduced from the large sizes of the fully-composed WFST to the small sizes as reported in Table 2.2. Moreover, as we will see in the next section, the compression techniques further shrinks the speech dataset (see Table 5.2). Finally, the miss ratio for a Token Cache of 512 KB is reduced from 15% to 11.5%, as we use the compact representation for the tokens proposed in [80].

Based on the above analysis, we have decided to reduce the sizes of the caches with respect to the acceleration design of Chapter 4, as we found it beneficial for the overall energy consumption and area usage. Table 5.1 shows the cache sizes for the UNFOLD (see Table 4.1 for comparing with the previous accelerator's configurations). Our caches still provide smaller miss ratios due to the large reduction in the memory footprint of the datasets, and their smaller sizes result in lower energy per access and area. In addition, we reduce the total size of Hash Table in spite of using

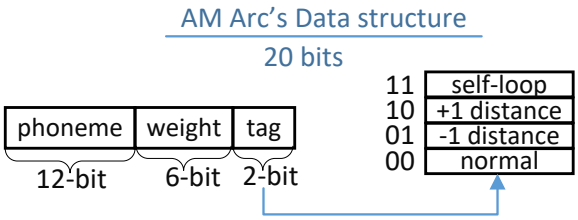


Figure 5.4: General structure of the AM arcs' information

the same number of entries, since the number of bits representing each of its entry's attributes is smaller using the compression scheme of Section 5.2.

Finally, Figure 5.3 shows the effect of the capacity of the Offset Lookup Table on the performance of the accelerator. As expected, the bigger the table, the smaller the miss ratio, resulting in higher performance in the accelerator. This table is direct-mapped, and it is indexed using the XOR of the LM state index and the word ID. Each entry contains a valid bit, a 24-bit tag and the 23-bit offset for the arc. By considering the area, energy per access and impact of the misses in the overall performance and energy consumption, we selected a table of 32K entries, which requires 192KB of storage.

5.2 WFST Compression

The use of on-the-fly composition significantly reduces the size of the datasets employed for ASR, as shown in Table 2.2. However, large vocabulary systems still require more than one hundred Megabytes for the AM and LM models. WFST compression can be applied on top of on-the-fly composition to further reduce the size of the datasets, achieving a large reduction compared to the original fully-composed WFST. This section describes the compression techniques employed in UNFOLD, including a description of the minor changes implemented in the hardware to support the compressed WFSTs.

The memory layout used in UNFOLD for each WFST is based on [21], which consists of two separate arrays for storing the states and the arcs information respectively. Regarding the states, we implement the bandwidth reduction scheme introduced in Section 4.3.2, that is also very effective for reducing the size of the states' information. On the other hand, arcs consume most of the storage of the WFST, since each state has hundreds or even thousands of outgoing arcs. Therefore, our compression techniques mainly focus on shrinking the size of the arcs.

Each arc consists of a 128-bit structure including destination state index, input label (phoneme index in AM and word ID in LM), output word ID and weight, each field represented by 32 bits. Regarding the weight, i.e. the likelihood of the arc, we use the K-means quantization technique with 64 clusters, reducing its size from 32 to 6 bits, which introduces a negligible increase in Word Error Rate (WER) of the on-the-fly ASR accelerator (less than 0.01%). For the other fields, we apply different techniques for LM and AM arcs.

Regarding LM data, we categorize the arcs in three different groups: the outgoing arcs of the

Table 5.2: Compressed sizes of WFSTs in Mega-Bytes for the fully-composed and on-the-fly composition.

Accelerator	Tedlium-EESEN	Tedlium-Kaldi	Voxforge	Librispeech
UNFOLD	39.35	32.39	1.33	21.32
Fully-composed	414.28	269.78	9.38	136.82

initial state (state 0 in Figure 2.10) or Unigram arcs, the back-off arcs and the rest. The Initial state consists of as many outgoing arcs as the number words of the vocabulary, each associated with a different word ID. As the destination states of these arcs are in consecutive order following their word IDs, their location can be inferred from the word ID. In other words, the i-th outgoing arc of state 0 is associated with word ID i and has destination state i. Therefore, no information other than a 6-bit weight value is required for each of these arcs.

On the other hand, back-off arcs in the LM model do not have any input or output label, they are only taken when no specific arc exists for a given word ID. In this case, we only store two attributes: the weight (6 bits) and destination state's index (21 bits). For the rest of the arcs, in addition to the aforementioned 27 bits, an 18-bit word ID is included to store the input label. We found that 18 bits is enough to represent every word of the vocabulary in all the tested ASR decoders. Note that, despite having variable length arcs, we still provide efficient random access to the LM information, which is a requirement of the binary search commented in Section 5.1. In our scheme, the back-off arc is always the last outgoing arc of each state, so it can be easily located in case the back-off mechanism is triggered. In addition, the rest of outgoing arcs of a given state have fixed size, taking 6 bits per arc for state 0 and 45 bits for the other states.

Regarding AM data, we have observed that, in our set of WFSTs, most of the arcs have epsilon word ID, i.e. they are not associated with any word in the vocabulary, and they point to the previous, the same or the next state. This is also the case in the example of Figure 2.10. For these arcs, we only store the input phoneme index (12 bits), weight (6 bits) and a 2-bit tag encoding destination state. This format is illustrated in Figure 5.4. The 2-bit tag indicates whether the arc points to the same state (self-loop arc), the previous (distance -1) or the next state (distance +1). The rest of the arcs require, in addition to the aforementioned 20 bits, an 18-bit word ID and a 20-bit destination state's index.

UNFOLD supports the compressed AM and LM models with some minor changes to the architecture. First, we include a 64-entry table (256 bytes) to store the floating-point centroids of the clusters used by the K-means approach to encode the weights of the arcs. The arc's information provides the 6-bit index of the cluster, and fetching the floating-point value requires an access to the table of centroids that is performed in an additional stage in the pipeline of the accelerator. Second, the Arc Issuer is extended to obtain the length of each AM's arc from the 2-bit tag, illustrated in Figure 5.4 . By decoding this tag, the Arc Issuer knows whether it has to fetch the remaining 38 bits for the current arc, or the 20 bits for the next arc. This serialization of the arcs does not introduce any additional complication, since the AM arcs of a given state are always explored sequentially in the Viterbi search. Third, the Arc Issuer is also extended to support the variable length arcs proposed for the LM model. The Address Generation Unit employs the state ID to obtain the size of the arcs (arcs departing from state 0 are packed in 6 bits, whereas outgoing arcs of other states have 45 bits). Note that the state's information provides the address of its first outgoing arc and,

hence, obtaining the address of its i-th arc is trivial once the size of the arcs is identified.

Table 5.2 shows the sizes of WFSTs used for the on-the-fly composition scheme after applying the compression techniques for AM and LM. As we can see, the dataset has been significantly reduced, from more than 1 GB or hundreds of MB (see Table 2.2) to a few tens of MB. Furthermore, we report the compressed size of the fully-composed WFST, for which we have implemented the compression techniques described in [81]. Compared to the compressed fully-composed WFST, our approach reduces the size of the required dataset by 8.8x on average. This large reduction is due to the combined use of both on-the-fly WFST composition and compression of the individual AM and LM models.

5.3 LAWS: Locality-aware Viterbi Expansion

In this section, we propose a new approach (LAWS) for implementing the Viterbi search in ASR systems that takes into account both the architectural and statistical features of the application in order to efficiently reduce the number of explored hypotheses with negligible impact on accuracy. First, by leveraging the locality among consecutive segments of the speech signal, LAWS exploits the on-chip recently-explored data while removing most of the off-chip accesses during the ASR's decoding process. As the second step, we introduce an approach to improve LAWS's effectiveness by selectively adapting the amount of ASR's workload, based on run-time feedback. In particular, we exploit the fact that the confidence of the ASR system varies along the recognition process.

In the following, we firstly go through the analysis of these two important features of ASR system, i.e. frame-to-frame locality and Viterbi search confidence, considering a significantly large speech test-bench. Then, we provide the detail on how we modify UNFOLD in order to employ these characteristics into the Viterbi search expansion.

5.3.1 ASR Feedbacks at Run-Time

In this part, we present the analysis on different properties of the recognition process in ASR. First, we evaluate the locality between the processing of successive frames of speech during the Viterbi search. Then, we define the basics of LAWS and illustrate its efficiency compared against a naive solution that consists on simply reducing the beam. Finally, we elaborate on our selective approach of dynamically changing the amount of ASR's workload using the Viterbi search's confidence.

ASR's Frame-to-Frame Data-Locality

In order to illustrate how useful data-locality is for the ASR's search expansion, we compute the average probability of the hypotheses, considering whether or not their data is on-chip. In general, probabilities are represented in the negative log-space for ASR systems, in order to simplify operations by replacing multiplications with additions and also prevent arithmetic underflows [110].

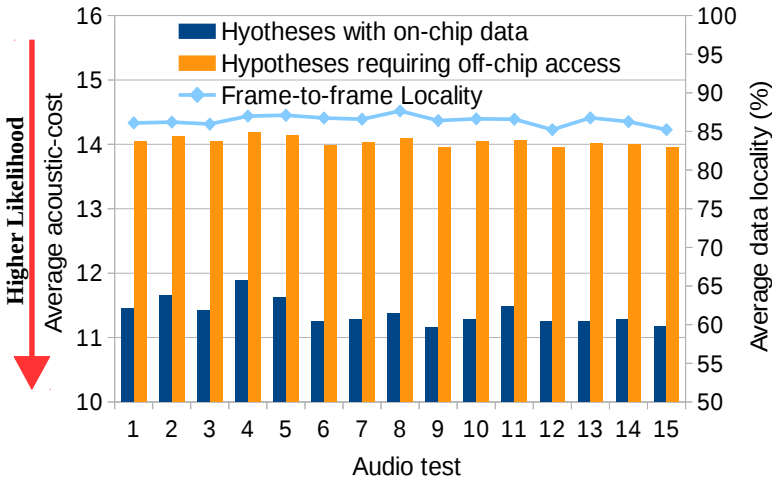


Figure 5.5: The bars show the average acoustic-cost of hypotheses grouped based on their data availability on-chip. We also shows the percentage of states and arcs that are the same in two consecutive frames (data locality).

Regarding the search pruning of Viterbi, as specified for the Kaldi ASR toolkit [76], we consider the beam of 15, representative as the very low probability of 3.1e-7. In other words, hypotheses whose distance to the best hypothesis up to the current frame is higher than 15 are discarded (pruned), in order to keep the search space tractable. Figure 5.5 shows the average acoustic-cost, i.e. log-space probability, of the different hypotheses, for 15 audio tests selected from different speakers of the Librispeech corpus [72]. In addition, the average locality of data is shown for each audio test, which shows nearly 86% of data-locality in processing subsequent frames. This means that, on average, 86% of the WFST's states and arcs used in a given frame are reused in the next frame. As depicted, hypotheses whose data is on-chip have considerably lower cost, i.e. higher probability, than the ones explored for the first time in each frame, whose data requires an off-chip memory access.

Furthermore, we extend our analysis to show that on-chip hypotheses are more likely to be in the correct path at the end of Viterbi search (full-hypothesis). We name the correct hypothesis per frame providing that it eventually ends up being part of the final answer of the Viterbi. To mark the correct hypotheses for all the speech frames, we initially perform the Viterbi search and store the information of all the hypotheses, then we process them backwards in order to locate the correct one. Considering the decoding of the entire Librispeech corpus (5.4 hours of speech), we have measured the likelihood of seeing the correct hypotheses regarding their data-status along the whole beam distance. Figure 5.6 depicts the probability of different hypotheses' being in the correct path with respect to the beam distance (distances are rounded up). Also, the probabilities are shown for the two groups of hypotheses based on whether their data is on-chip or off-chip. As seen, most of the correct hypotheses are more likely in the distance of less than 1 to the best one. Moreover, the hypotheses using on-chip data have three to four orders of magnitude higher chance of being in the correct path.

Based on our evaluations, we propose to prioritize the expansion of hypotheses that exhibit good temporal locality, since they are less expensive from a computational point of view, which is beneficial for performance and energy, whereas they are more likely to be in the correct path, which

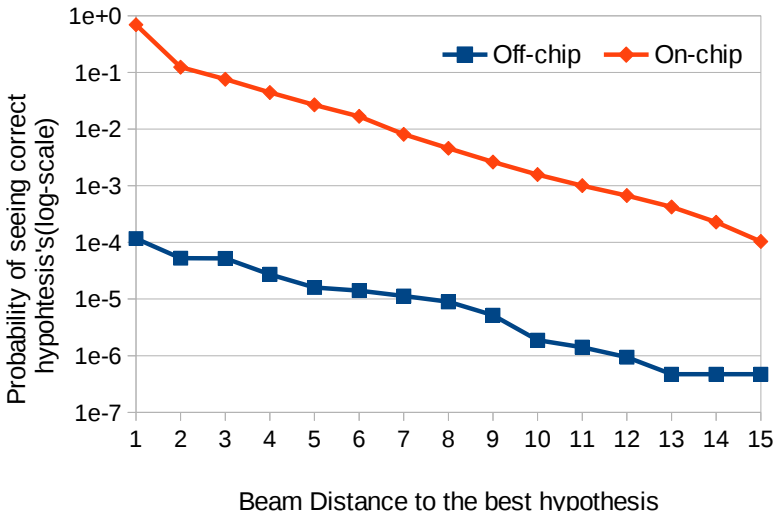


Figure 5.6: The diagram shows the probability of seeing correct hypotheses in different beam distances. The distances are rounded up. Also, the likelihoods are shown based on whether or not hypotheses' data is on-chip.

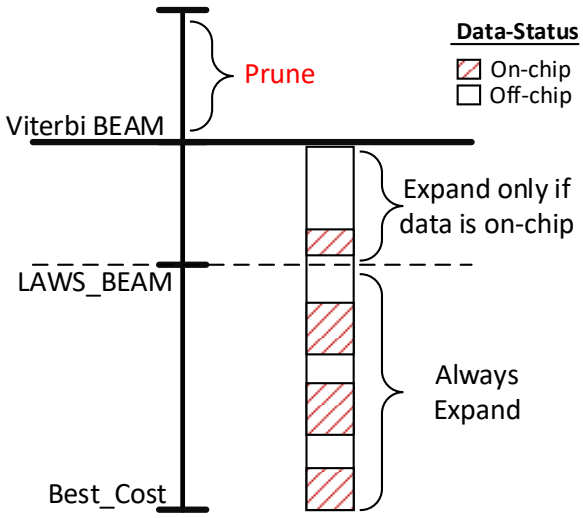


Figure 5.7: Viterbi expansion under Locality-AWare Scheme.

is beneficial for accuracy. On the other hand, we are more selective with the hypotheses that require off-chip memory accesses, using a more aggressive beam. Figure 5.7 illustrates the Viterbi search expansion for both the conventional beam search and our Locality-AWare Scheme (LAWS). In the conventional approach, the hypothesis with the best cost is identified on every frame. Hypotheses whose distance to the best hypothesis is smaller than a given beam are explored, whereas the rest are discarded since they are very unlikely. Our scheme is different since we consider both the cost (likelihood) of the hypotheses and their temporal locality to prune the search space.

LAWS defines an additional beam, named LAWS_BEAM in Figure 5.7, smaller than the original beam. Hypotheses whose distance to the best hypothesis is smaller than the LAWS_BEAM are

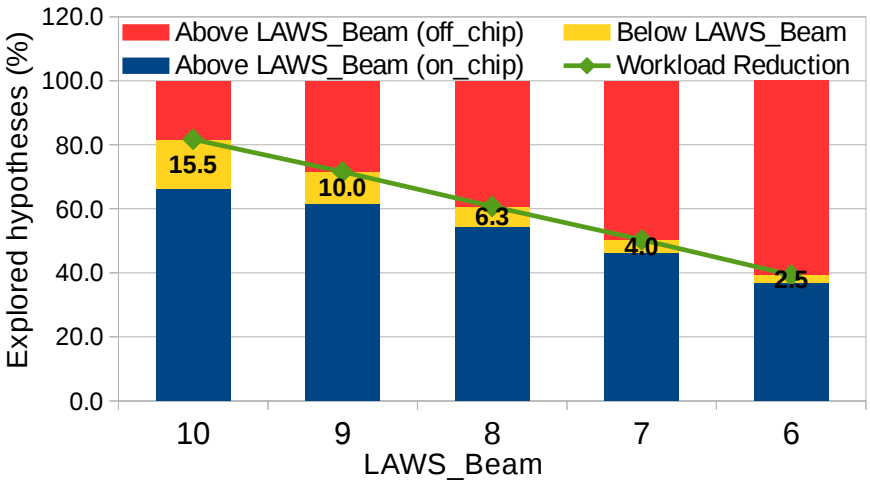


Figure 5.8: Hypotheses distribution along the LAWS_Beam and Viterbi beam distance. Hypotheses whose cost are above LAWS_Beam are partitioned into on-chip and off-chip, showing the status of their data. By removing off-chip hypotheses above LAWS_Beam, we can have the workload decrease as shown in the graph.

still expanded, since the likelihood that they end up in the correct path is significant. Hypotheses whose distance is larger than the original beam are still discarded as in the conventional search, since they are very unlikely. However, hypotheses between LAWS_BEAM and the Viterbi beam are only explored if their data is on-chip. Note that those are hypotheses that are always explored in the conventional search, but they are not very likely to be in the correct path. Therefore, we consider that the cost of accessing main memory is excessive for such unlikely hypotheses and, hence, we discard them. Figure 5.8 shows that a significant percentage of the hypotheses are above the LAWS_BEAM and have their data off-chip and, hence, our scheme is effective at reducing workload. More specifically, LAWS is able to save between 20% and 60% of the Viterbi search workload depending on the selection of the LAWS_BEAM.

Note that our scheme is better than a naive solution that just lowers the beam used in the Viterbi beam search, since hypotheses whose distance is between the LAWS_BEAM and the Viterbi beam are still explored provided that their data is on-chip. Furthermore, exploring those hypotheses is cheap since they do not require off-chip memory accesses. Hence, from the point of view of main memory, our scheme is equivalent to a conventional Viterbi beam search with a more aggressive beam, but we achieve much better accuracy since we explore more hypotheses using already available on-chip data. Figure 5.9 shows the ASR accuracy, i.e. Word-Error-Rate (WER), for decoding the entire Librispeech corpus, using different values for the Viterbi beam and the LAWS_BEAM. As we can see, lowering the beam has a large impact on the accuracy of the conventional Viterbi search, whereas LAWS is able to use more aggressive beams with a smaller impact on WER, since the aggressive beam is only used for hypotheses with bad temporal locality, i.e. whose data is not on-chip. In the next subsection, we show that by changing the LAWS_BEAM dynamically, we can further reduce the impact on accuracy while obtaining large performance and energy improvements.

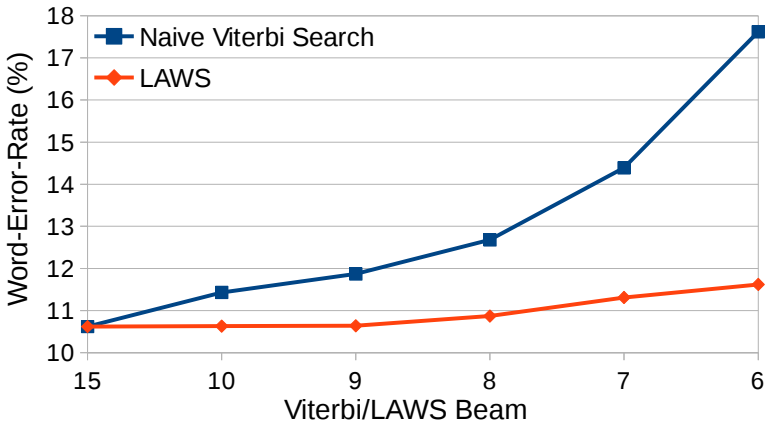


Figure 5.9: Word-Error-Rate (WER) change with respect to different beams selected for LAWS and naive Viterbi search.

Improving LAWS Using Viterbi's Confidence

LAWS is highly effective at saving energy consumption and main memory bandwdith by significantly reducing ASR's workload. However, the main issue is the increase in WER which can be non-negligible for some applications that require a high level of accuracy. In order to handle this problem, we analyze the influence on the recognition accuracy by changing the amount of workload toward different segments of the speech signal. We evaluate the impact on the selection of correct hypotheses under the confidence of search at different frames. We show that, by receiving the feedback of search confidence at each frame, we are able to define an adaptive approach in dynamically selecting the amount of workload to improve ASR's accuracy.

In addition to taking into account the locality of the data, besides the hypotheses' likelihood scores, LAWS uses the confidence of the ASR system at each frame. There are some frames of speech for which there are many hypotheses with scores very close to the best, whereas there are fewer in other frames. When the number hypotheses close to the best is high, we say that the ASR system has low confidence, since there are many alternatives similar to the best, whereas we say that the confidence is high when the number is low.

Figure 5.10 shows an example of how the number of hypotheses that are close to the best one (using the default fixed beam in Kaldi toolkit) changes during the evaluation of the frames of one sample audio test. We can observe that there are huge variations. For some frames, we have tens of thousands of hypotheses, whereas for others, we just have a few tens. LAWS exploits this information to adapt the beam search. We have seen that when the confidence is high, the correct hypothesis tends to be closer to the best one than when the confidence is low. This suggest that for high confidence intervals, we can be more restrictive with the beam to save energy without negatively impacting accuracy.

Viterbi search exhibits different levels of confidence when decoding different frames of speech. In other words, frames exhibit a high variation in the amount of workload. Therefore, in order to have an effective approach to decide the degree of pruning based on the confidence of search, we need to efficiently categorize the frames of speech in different audio tests. To do so, we partition

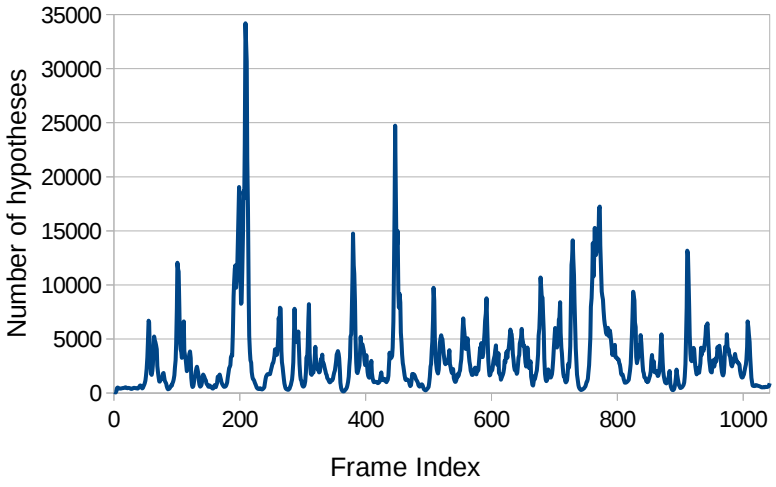


Figure 5.10: The number of hypotheses along the frames of an audio test containing 1024 frames (10.42 seconds of speech) for the default fixed beam used by Kaldi.

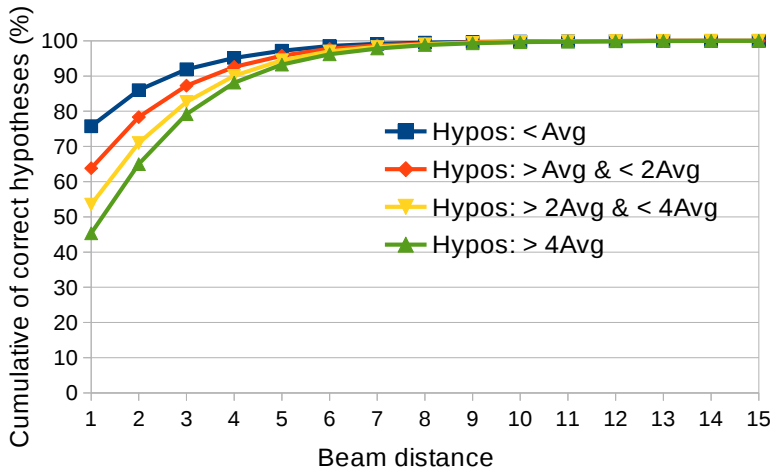


Figure 5.11: Cumulative percent of the correct hypotheses explored under different beam distances specified in Viterbi search, for the different group of frames based on their generated hypotheses.

audio frames in a way that each group has approximately the same amount of work during the Viterbi search. We do that to ensure that all groups of frames have the same importance as defined by the number of their hypotheses. Then, we measure the impact in accuracy by changing the beam at each group to validate our hypothesis that when the number of hypothesis is large, confidence is low and thus correct hypotheses tend to be farther from the optimal score (in other words, a larger beam should be used), and vice versa. According to our analysis on the Librispeech corpus, we can control the workload of different regions of frames using the dynamically-computed average number of hypotheses explored in all frames. We use several multiples of the computed average to distribute the hypotheses among different groups uniformly as shown in Figure 5.12. We define four categories to simplify our technique. However, more groups can be used to have finer granularity in controlling the ASR workload.

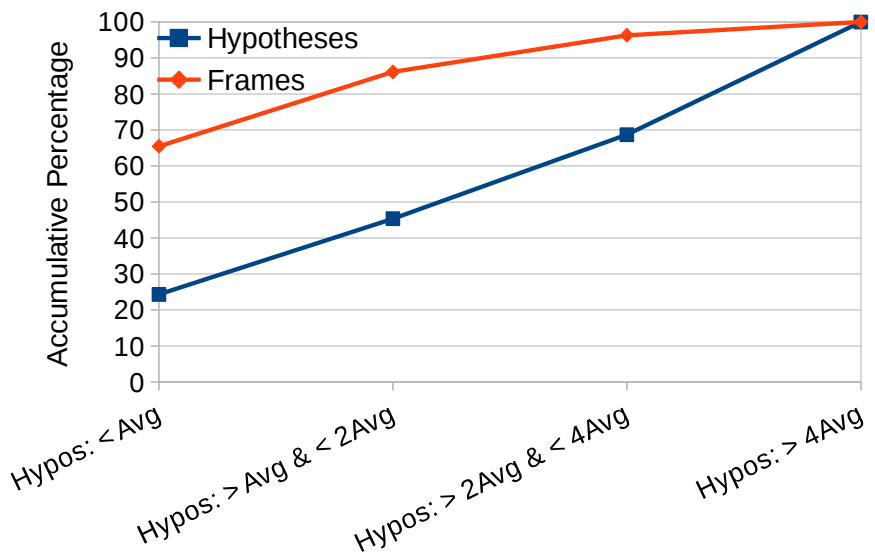


Figure 5.12: cumulative percentage of hypotheses and frames explored during entire 5.4 hours of audio in Librispeech corpus, for different groups of frames based on the number of hypotheses.

The impact of Viterbi's beam pruning varies for each region of the speech frames due to the difference in their confidence. In order to analyze such difference in sensitivity, we compare the number of correct hypotheses that is expanded under a certain beam distance for the different groups. Figure 5.11 shows the cumulative percentage of correct hypotheses expanded for the frames' partitions separately, considering the range of beam distances between 1 and 15 (distances are rounded up discretely). As illustrated, at each certain distance from the best hypothesis, higher percentage of correct hypotheses can be evaluated for the group of frames with higher confidence. On the other hand, if we want to reach a certain level of accuracy for the different groups, we need to choose a range of beam values assigning them based on the confidence of search at each frame. As depicted, the higher the confidence of the frames, the smaller the distance of beam is required.

As illustrated in Figure 5.12, various regions encompass uneven percentages of frames, in contrast with the uniform distribution of the hypotheses between different groups. As seen, nearly 65% of the frames have the number of hypotheses less than average. On the other hand, only few percentage of frames (5%) have hypotheses-count higher than 4 time of average. Consequently, due to the occurrence of large number of frames in the partitions with high confidence, we can improve the ASR's accuracy by selecting a more conservative beam distance in these regions. On the other hand, we compensate that by reducing more the beam in the frames which are low-confident and have more number of hypotheses, reducing more the ASR's workload. Therefore, although Figure 5.11 suggests that low-confident frames require higher beam distance than the high-confident ones in order to reach an accuracy level, we can refine that in a way to obtain better energy and memory savings.

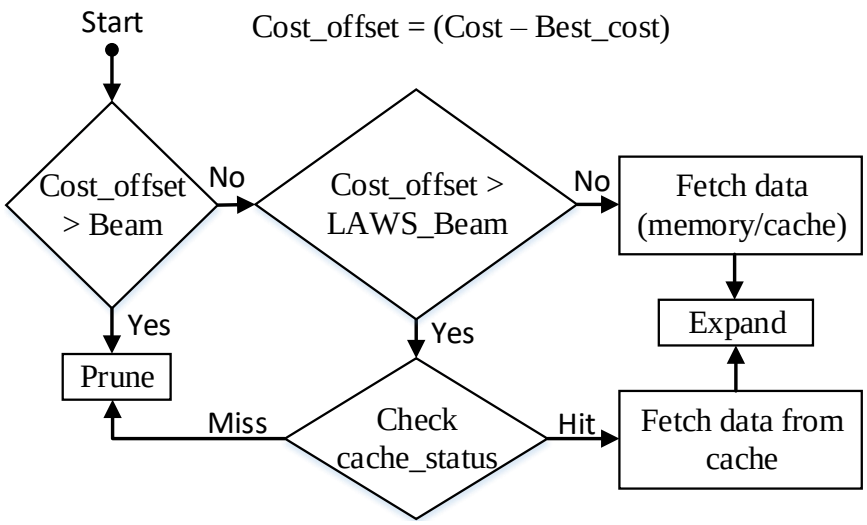


Figure 5.13: Flow-chart of LAWS for the Viterbi expansion.

5.3.2 Implementing LAWS Mechanism for UNFOLD's Architecture

In here, we present the implementation of LAWS on top of UNFOLD. Then, we provide some exploration to show the benefits of the approach. Next, we evaluate different strategies in order to specify the pruning beam based on the search confidence. Finally, we introduce several configurations by combining our proposed techniques to provide an efficient trade-off between accuracy, performance and energy of the ASR system.

LAWS Mechanism

The analysis of section 5.3.1 shows the benefit of LAWS in efficiently expanding the Viterbi search of ASR systems. Unlike the previous schemes, our approach considers both the likelihood of the evaluated hypotheses and the locality of their data on the on-chip memory components as an important aspect for driving the search. LAWS's mechanism is included in the State Issuer of the UNFLOD's architecture, after fetching each hypothesis' information from the Hash table of the current frame (see Figure 5.1). Figure 5.13 shows the flow-chart of LAWS for the Viterbi search expansion. As illustrated, a second comparison is added after passing the first beam pruning of Viterbi, which compares the hypothesis's cost-offset (cost difference with the best-cost) with the LAWS_Beam. Then, if passing the second pruning condition, the data is fetched either from memory or cache, otherwise the state of cache in the accelerator's design is checked. If the data is available in cache, the hypotheses is retrieved and expanded, otherwise we prune that. We add a snoop port in the State Cache that is used to check whether the data of the hypothesis is available on-chip without interfering with other operations performed in the accelerator.

In order to further explore our proposal, we implement it on top of UNFOLD and evaluate the Viterbi search execution time. Figure 5.14 shows the speedup achieved in the performance of Viterbi with respect to the baseline (LAWS_Beam = 15) using different LAWS beam values. As

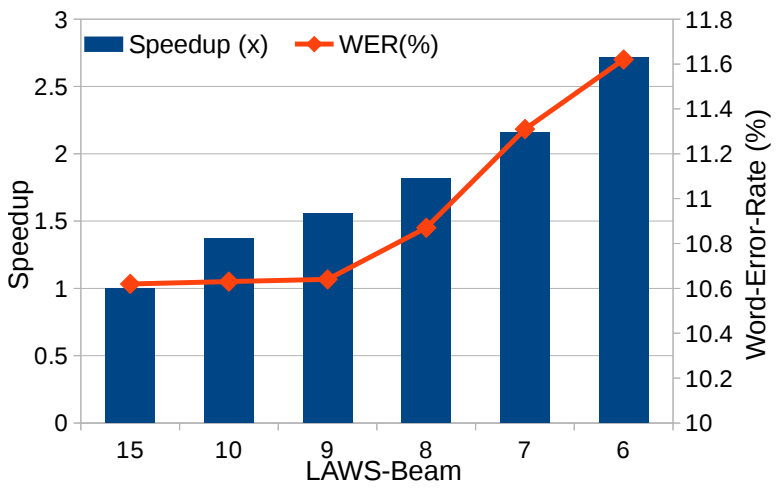


Figure 5.14: Speedup and Word-Error-Rate (WER) obtained for LAWS using different LAWS_Beam. LAWS achieves 55% speedup without losing any accuracy. Further performance improvement causes some accuracy loss.

seen in this figure, we can obtain nearly 55% performance improvement over UNFOLD by using LAWS_Beam of 9 without losing any accuracy. However, if higher speedup is required, ASR loses some accuracy up to 1% when using LAWS_Beam 6. To solve this issue, we combine LAWS with a new method of controlling ASR accuracy, by dynamically adapting the beam during runtime of Viterbi based on the confidence of the search in various frame's evaluation.

Size of *State Cache* is an important aspect when combining hypotheses' likelihood with their data-locality. Thus, we have tested several cache sizes to measure how performance differs by considering various LAWS-beam widths. Figure 5.15 shows the speedup achieved using each cache size versus different LAWS beams. As illustrated, using a bigger cache than UNFOLD's base configuration would result in approximately similar speedup for LAWS. On the other hand, by decreasing the cache size, we see that speedup can grow significantly by applying a more aggressive LAWS beam width. However, these improvements come at the cost of losing some accuracy because some correct hypotheses are discarded due to their poorer locality.

Adaptive Beam-Selection Approach

As discussed in Section 5.3.1, the confidence of Viterbi search is one of the important ASR's characteristics in defining the workload amount at different frames. To measure such feedback during the search, we count the number of hypotheses that are needed to expand at each frame. Using this number, we can score the confidence of each frame with respect to the different confidence regions specified by the four groups of frames. As aforementioned, by choosing a range of beam distances rather than just one for the different regions, we can obtain higher accuracy while significantly reducing the ASR's workload. Therefore, we define a multi-beam selection scheme for LAWS that can adapt the LAWS_Beam dynamically based on the search confidence.

Figure 5.16 shows the different models of selecting the beam distance for managing the ASR's

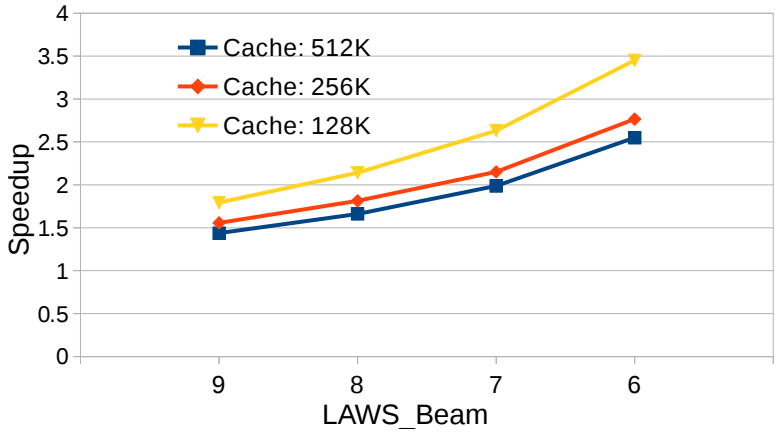


Figure 5.15: LAWS speedup versus beam width, for different sizes of cache. The larger the cache, the higher the locality it provides and the more hypotheses are explored, resulting in lower speedup.

workload based on the search confidence. The Single-beam scheme considers only one beam disregarding the confidence of the different frames' evaluation. However, as mentioned in Section 5.3.1, we are limited to almost 30% of workload reduction using one LAWS_Beam in order to maintain ASR accuracy. On the other hand, the Multi-beam model suggests a highly flexible approach of choosing the beam values proportional to the search confidence. Nevertheless, we have seen that we can obtain most of the benefits of this scheme by having two or three beam distances. In the last model, called Dual-Beam, we choose between the two high and low beams for the low and high confident regions, respectively. Finally, in order to gain more reduction in ASR's workload, we have tuned the relation between confidence and beam distance in a way to adopt some accuracy loss by decreasing the beam in the low-confident regions. Moreover, we compensate for the accuracy loss using higher beam in the high confident regions. Using such tuning, we can achieve higher benefits for smaller cost in accuracy, since the regions with high confidence include significantly higher percentage of frames (see Figure 5.12).

As the Dual-beam approach is simple and highly efficient, we implement it in the Viterbi accelerator to improve LAWS's accuracy. We choose 6 as the low LAWS_Beam since it provides highest reduction, and 8 as the high LAWS_Beam to compensate for the accuracy with negligible loss compared to the baseline. We define several thresholds on the search confidence in order to decide for the LAWS_Beam during the run-time of Viterbi. We call our management technique as $Small_8$-Big_6, which selects the beams 8 and 6 for the frames with the number of hypotheses smaller and bigger than a threshold, respectively.

Figure 5.17 shows the way $Small_8$-Big_6 performs considering different confidence thresholds (defined as multiples of average number of hypotheses). Furthermore, we show the result of the single-beam approach considering several beam widths. As depicted, the single-beam scheme loses between 0.7% to 1% accuracy to achieve the same performance as $Small_8$-Big_6. On the other hand, $Small_8$-Big_6 obtains a speedup of 2.45x with negligible increase of 0.32% in WER. As illustrated, by using beam 6 at more regions with low confidence, we can obtain better speedup while losing some accuracy. Therefore, based on each application's restriction, we can choose each of these thresholds to both maintain the accuracy and gain high performance benefits. Furthermore, we have used

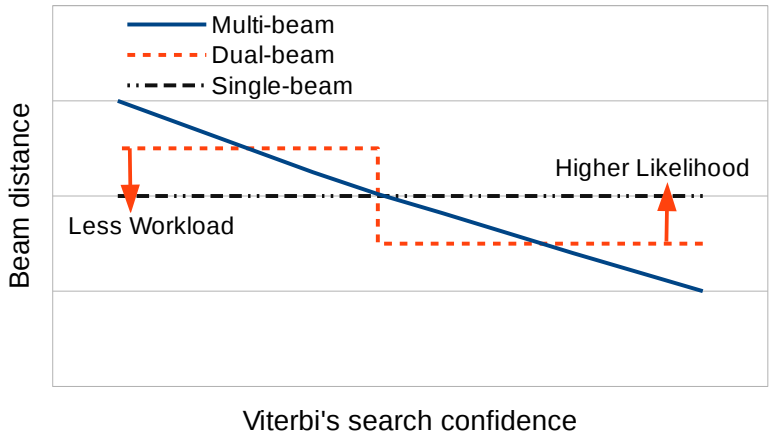


Figure 5.16: Different beam selection models based on the confidence of Viterbi search, i.e. number of hypotheses. Single-beam uses one beam independent to the search confidence. The multi- and dual-beam models choose from the big to small distances according to the low to high levels of confidence in search, respectively.

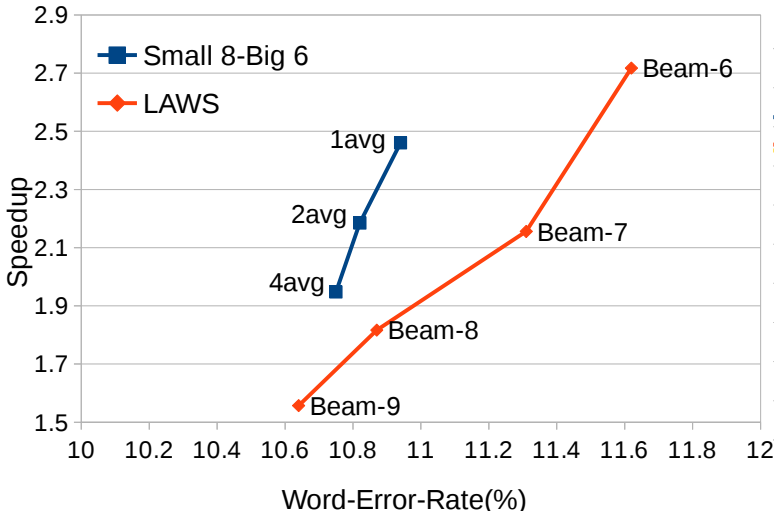


Figure 5.17: Speedup versus Word-Error-Rate (WER), for the Dual-beam selection scheme, called $Small_8$-Big_6. High confidence threshold provides better speedup, whereas maintaining accuracy.

several configurations using three LAWS_Beams to improve the trade-off between accuracy and workload reduction. However, we experience some slight improvement in speedup as well as losing some accuracy. Therefore, we conclude to use only two beams as it results in a simpler scheme.

5.4 Experimental Results and Analysis

In this section, we firstly evaluate UNFOLD, our proposed hardware accelerator based on the on-the-fly WFST composition. Doing so, we analyze the reduction in memory footprint achieved by UNFOLD, we review its energy consumption and performance, providing a comparison with

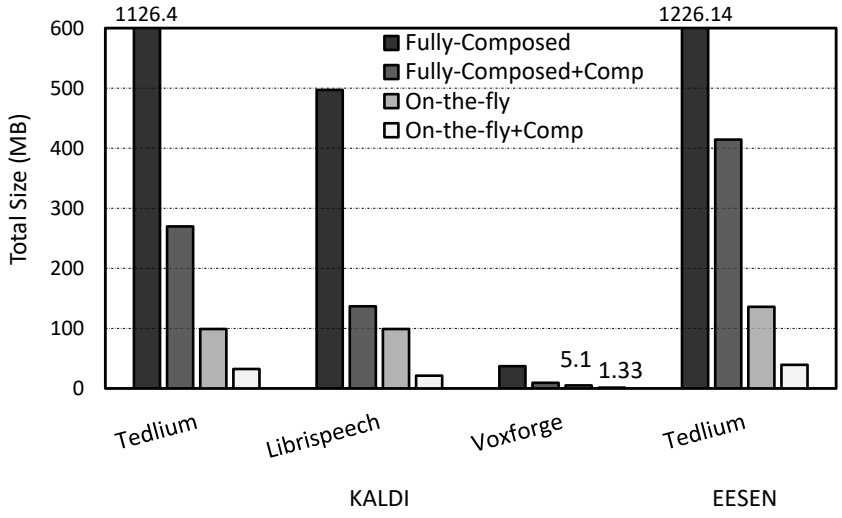


Figure 5.18: Sizes of different datasets for several ASR systems.

both our previous accelerator and a solution based on a contemporary mobile GPU. Second, we evaluate the benefits of LAWS, by evaluating the performance speedup and energy-saving obtained for the UNFOLD baseline. Furthermore, we present the reduction in the memory requirements gained by exploiting the locality in ASR.

5.4.1 UNFOLD's Evaluation

In this part, we initially focus on the acceleration benefits for the Viterbi search of the ASR's pipeline, comparing UNFOLD's benefits with respect to the previous designs. Next, we evaluate the entire ASR pipeline, including UNFOLD and the GPU for running the GMM/DNN/RNN.

Viterbi Search Acceleration

Figure 5.18 shows the memory requirements of the WFSTs for the fully-composed and on-the-fly composition approaches in several ASR systems. *Fully-Composed* configuration represents the sizes of the WFSTs for the accelerator presented in Chapter 4. *Fully-Composed+Comp* implements the compression techniques introduced in [81] for the fully-composed WFSTs. *On-the-fly* shows the total size of the individual AM and LM models, without any kind of data compression. *On-the-fly+Comp* shows the memory requirements for UNFOLD, which includes on-the-fly WFST composition and the compression techniques described in Section 5.2.

As it can be seen in Figure 5.18, the *On-the-fly* approach provides consistent savings in memory requirements in the different ASR decoders. On-the-fly composition mechanism avoids the explosion in the number of states and arcs resulted from the full offline-composition of AM and LM models. Furthermore, the compression techniques provide significant additional savings. Overall, UNFOLD

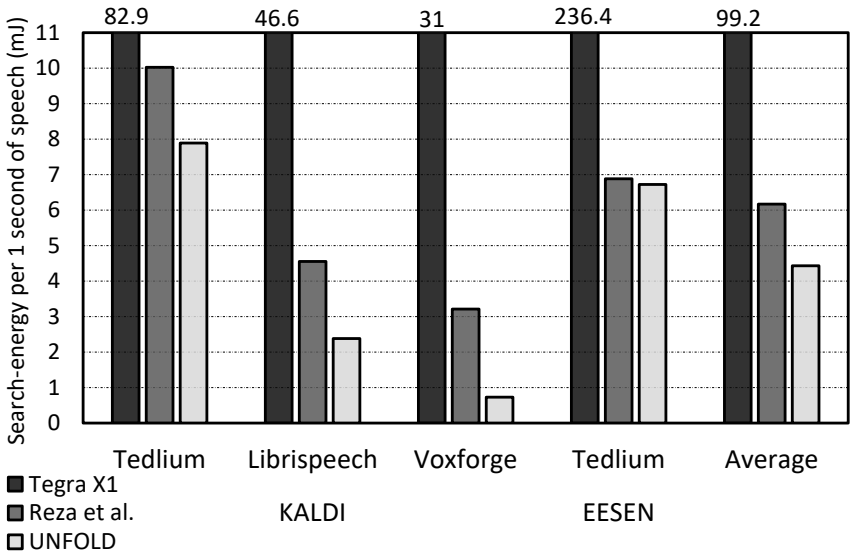


Figure 5.19: Viterbi search energy consumption of several speech recognizer in different platforms.

achieves a reduction in storage requirements of 31x on average with respect to the state-of-the-art Viterbi search accelerator discussed in Chapter 4 (*Fully-Composed configuration*), with a minimum and maximum reduction of 23.3x and 34.7x respectively. With respect to the state-of-the-art on WFST compression [81] (*Fully-Composed+Comp configuration*), our system achieves a reduction of 8.8x on average. All in all, UNFOLD only requires a few tens of Megabytes, which is reasonable for many mobile devices, unlike the more than one Gigabyte required by previous proposals.

Figure 5.19 shows the energy consumption for the Viterbi search, including both static and dynamic energy, in several ASR systems. The configuration labeled as *Tegra X1* is the CUDA-based solution running on the mobile SoC. *Reza et al.* configuration is the Viterbi accelerator proposed in Chapter 4. Finally, the energy of the proposed accelerator is shown in the configuration labeled as *UNFOLD*. *UNFOLD* reduces energy consumption by one order of magnitude with respect to the *Tegra X1*. Compared to the state-of-the-art in Viterbi search acceleration, it provides consistent energy savings in all the ASR decoders that range between 2.5% for EESEN-Tedlium and 77% for KALDI-Voxforge, with an average energy reduction of 28%. UNFOLD generates significantly less requests to main memory since the caches exhibit larger hit ratios, as reported in Section 5.1.3, due to the large reduction in the size of the datasets. On average, *UNFOLD* reduces off-chip memory accesses by 68% with respect to the *Reza et al.* configuration. Therefore, although it requires more on-chip activity to implement on-the-fly composition and handle the compressed datasets, it provides important energy savings due to the reduction in energy-expensive off-chip memory accesses [42].

Figure 5.20 shows the power breakdown, including both static and dynamic power, of the *Reza et al.* and *UNFOLD*. As we can see, power savings are mainly due to the reduction in the power dissipation of main memory, caused by the aforementioned savings in off-chip memory traffic. The power of the caches is also reduced, as they have smaller sizes (see Table 5.1) since our datasets are smaller. On the other hand, the power of the Token Cache is reduced by a large extent since we include the compressed format for the word lattice proposed in [80]. In the hash tables, as we

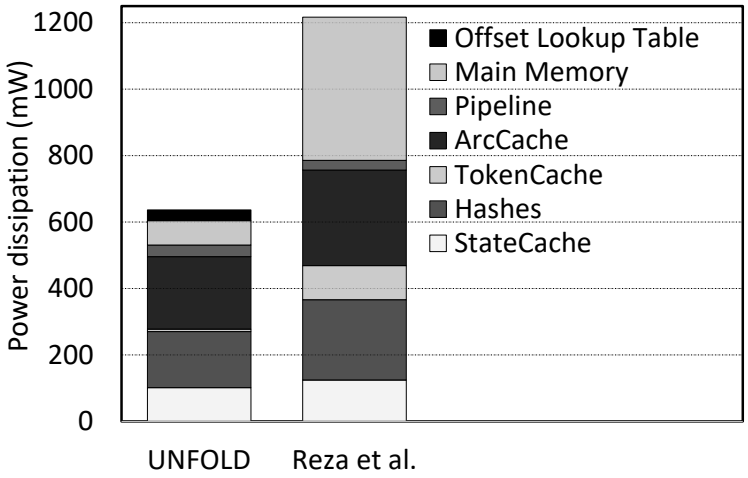


Figure 5.20: Power breakdown of UNFOLD versus Reza et al.

are reducing the size of the attributes, the storage requirements are smaller compared to the fully-composed approach, and this results in less power dissipation. Finally, the *Offset Lookup Table*, which is required for efficiently locating LM arcs in our system, represents a small overhead since it only dissipates 5% of the total power in UNFOLD.

Figure 5.21 shows the memory bandwidth requirements of both fully-composed and on-the-fly accelerators when running different ASR decoders. As it can be seen, UNFOLD provides significant and consistent memory bandwidth savings for all the decoders. For the most bandwidth demanding decoder, EESEN Tedlium, UNFOLD reduces the memory bandwidth usage by nearly 2.8x, from 7.4 to 2.6 GB/s. On average, UNFOLD reduces memory bandwidth usage by 71%. These savings are mainly due to the large reduction in the size of the datasets (see Figure 5.18) provided by the on-the-fly composition and compression mechanisms. The smaller size of the datasets produces a large reduction in the number of cache misses, as described in Section 5.1.3, which results in an important reduction in the number of main memory accesses.

Regarding the decoding time, all the configurations achieve real-time performance by a large margin, which is the main requirement of ASR systems. *Tegra X1* runs 9x faster than real-time, whereas *Reza et al.* and *UNFOLD* run 188x and 155x faster than real-time respectively. Table 5.3 shows an analysis of the processing time per utterance for the two accelerators and the mobile GPU, including the average and the maximum processing time. Processing time per utterance affects to the responsiveness of the system and overall user experience. As it can be seen, both accelerators achieve significantly lower processing time than the mobile GPU, providing the result within tens of milliseconds on average and a few hundreds of milliseconds in the worst case.

The slowdown in our system with respect to the *Reza et al.* is due to the cost of fetching the arcs in the LM model, which requires a search across the outgoing arcs of a state to find the arc associated with a given word ID. By replacing the simple linear search with a binary search, we reduce this slowdown from 10x to 3x. Furthermore, the use of the *Offset Lookup Table* and the preemptive pruning (see Section 5.1.2) reduce this slowdown to an average of 18% with respect to the state-of-the-art in Viterbi search acceleration. This overhead represents an increase in decode

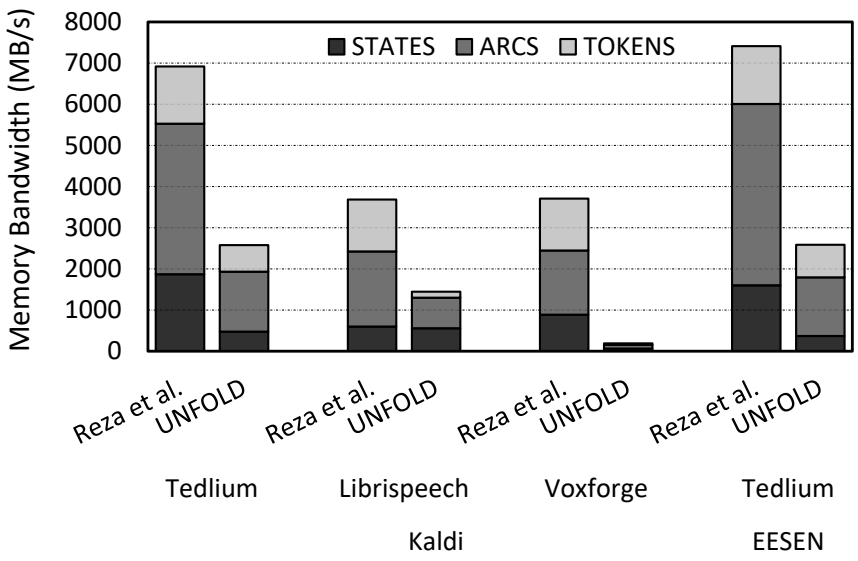


Figure 5.21: Memory bandwidth usage of Reza et al. and UNFOLD for different ASR decoders.

Table 5.3: Maximum and average decoding time (ms) per utterance for different decoders and platforms.

ASR Decoder	**Tegra X1**		**Reza et al.**		**UNFOLD**	
Timing Parameter	**Max**	**Avg**	**Max**	**Avg**	**Max**	**Avg**
Tedlium-Kaldi	2538	1069	215	76.7	274	92.5
Librispeech	1651	1336	54.7	31.9	57.4	30
Voxforge	743	450	47.1	15.5	13.1	4.2
Tedlium-EESEN	3972	1412	222	60	680.4	111.6

Table 5.4: Word Error Rate (WER) of UNFOLD for different ASR decoders.

ASR Decoder	**Tedlium-Kaldi**	**Librispeech**	**Voxforge**	**Tedlium-EESEN**
WER(%)	22.59	10.62	13.26	27.72

time for a second of speech from around 5 ms to 6 ms, which is practically negligible for typical applications, since it keeps being two orders of magnitude faster than real-time.

In order to evaluate the accuracy of UNFOLD, our cycle-accurate simulator also works as a functional emulator able to produce the most likely sequence of words for the input speech. Table 5.4 shows the Word Error Rate (WER) for the different ASR decoders. Note that the GMM/DNN/RNN and the WFST are trained on hundreds of hours of audio from the different training datasets: Tedlium, Librispeech and Voxforge. The WER is reported for the entire test datasets that include several hours of speech. TEDLIUM is a more challenging task as it includes spontaneous speech in a noisy environment and, hence, the WER is larger than in Librispeech or Voxforge. Furthermore, we have verified that the difference in accuracy with respect to the fully-composed approach is negligible, i.e. the compression and on-the-fly mechanisms do not introduce any noticeable accuracy loss.

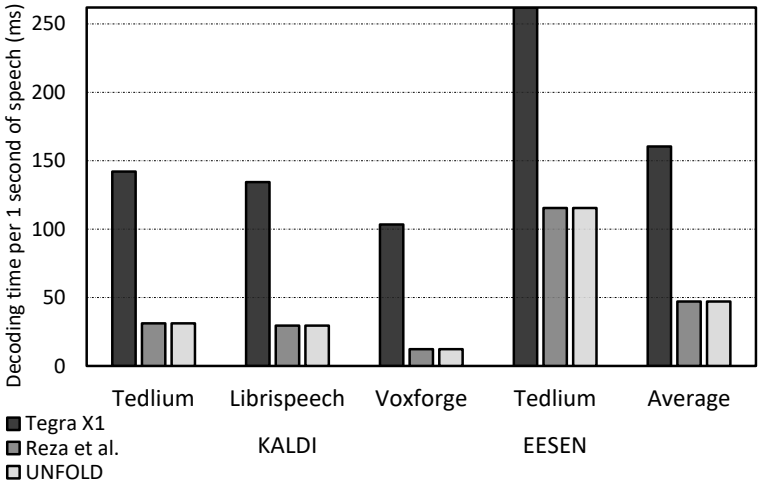


Figure 5.22: Overall ASR system decoding-time in different platforms.

Regarding the area, UNFOLD has a size of 21.5 mm 2 , providing a reduction of 16% with respect to the Reza et al. configuration. This reduction is due to the smaller caches used, as described in Section 5.1.3.

To sum up, the UNFOLD's on-the-fly WFST composition scheme provides a reduction in memory storage requirements of 31x and 28% energy savings on average with respect to the state-of-the-art in Viterbi search acceleration. On the other hand, it introduces a slowdown of 18% and a small increase in Word Error Rate of less than 0.01% due to the clustering of the weights. We believe that the important savings in memory footprint and energy consumption largely compensate for the negligible slowdown and accuracy loss. In other words, using 40 Mbytes of memory instead of more than one Gigabyte makes a huge difference, and saving 28% energy is very important, but being 155x faster than real-time instead of 188x has no impact on user experience.

Overall ASR System

In this section we evaluate the memory requirements, performance and energy consumption for the overall ASR system. In addition to the Viterbi search, the KALDI-Tedlium and KALDI-Voxforge decoders employ a GMM for acoustic-scoring. The KALDI-Librispeech employs the DNN presented in [115] for acoustic scoring. Finally, the EESEN-Tedlium uses a Recurrent Neural Network (RNN) [63] to generate the likelihoods consumed by the Viterbi search.

Regarding the sizes of the datasets, our system provides a compression ratio of 15.6x on average when considering both the WFST and the GMM/DNN/RNN. After applying on-the-fly composition and WFST compression, the GMM/ DNN/RNN represents approximately half of the dataset for each of the ASR systems evaluated except for the Voxforge whose GMM takes 26% of the storage required for ASR.

Figure 5.22 shows the decoding time per one second of speech for different configurations. *Tegra X1* configuration runs the entire ASR pipeline on the mobile GPU. *Reza et al.* and *UNFOLD* exe-

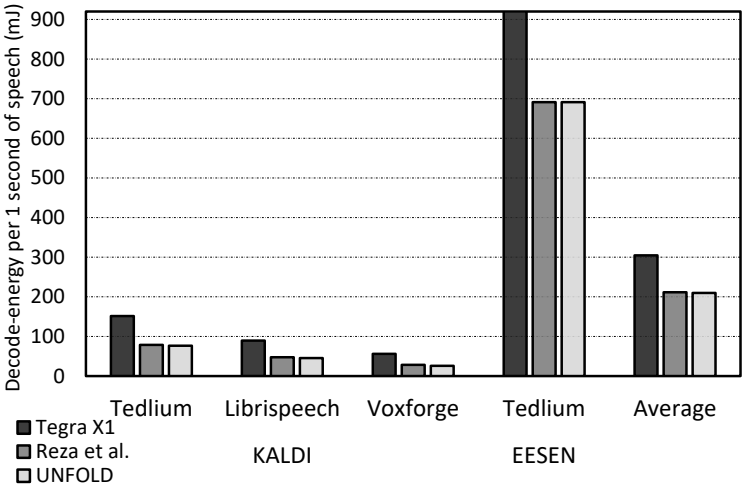


Figure 5.23: Overall ASR system energy consumption in different platforms.

cute the Viterbi search on the accelerator proposed in fully-composed and our on-the-fly accelerator respectively, while the GMM/DNN/RNN is executed on the mobile GPU. The integration between the GPU and the accelerator is performed as described in Section 4.1. In the overall system, the input speech is split into batches of N frames and the GPU and the accelerator work in parallel: the GPU computes the acoustic scores for the current batch while the accelerator performs the decoding of the previous batch. The GPU communicates the acoustic scores through a shared buffer in main memory, and we account for the cost of these communications in our performance and energy numbers for the overall ASR system. All the decoders achieve real-time performance, and the configurations using hardware-accelerated Viterbi search achieve similar decoding time, outperforming the GPU-only configuration by approximately 3.4x.

Figure 5.23 shows the energy consumption per one second of speech for the same configurations. *Reza et al.* and *UNFOLD* provide similar energy reduction of 1.5x over the *Tegra X1* configuration. The reason why these two configurations exhibit similar behavior is that, after accelerating the Viterbi search, the GMM/DNN/RNN becomes the most time and energy consuming component, as it runs in software on the GPU. Note that, if further energy savings or performance improvements are required, an accelerator for the GMM [99], DNN [87] or RNN [92] can be employed. Accelerating these algorithms is not the scope of this paper, we focus on the Viterbi search as it is the main bottleneck in state-of-the-art ASR systems.

In summary, when considering the entire ASR pipeline, our system achieves similar performance and energy consumption to the state-of-the-art proposals, but providing a reduction of 15.6x in the memory storage requirements. Our system requires less than 80 Megabytes for the entire ASR datasets in the worst case, whereas previous proposals consume more than one Gigabyte.

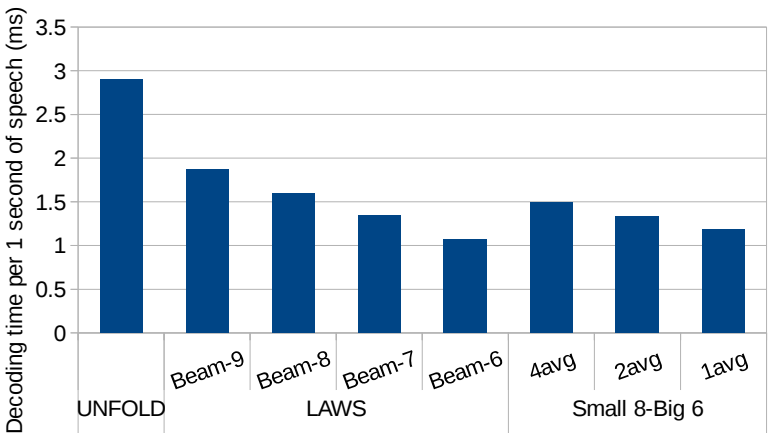


Figure 5.24: ASR's decoding time per one second of speech for the baseline and different configurations of LAWS using single and dual beams.

5.4.2 LAWS's Evaluation

After completely evaluating UNFOLD's benefits in ASR acceleration, we focus on its further improvement using LAWS mechanism. We report the performance speedup and energy-saving with respect to the baseline UNFOLD's results. Furthermore, we will show how LAWS significantly reduces the memory requirements of the Viterbi accelerator. In the next chapter, we will focus on DNN acceleration since this part becomes the main bottleneck after thoroughly improving the Viterbi search phase. Moreover, we will discuss the interaction between the two accelerators in the following chapter.

Figure 5.24 shows the Viterbi and DNN breakdown of the ASR's decoding time per one second of speech for the different approaches. We refer to our baseline design and the different configurations of LAWS as UNFOLD, and LAWS-Beam-8, LAWS-Beam-7, and LAWS-Beam-6, respectively. Also, we use analogous terminology for $Small_8$-Big_6 scheme using the different confidence thresholds in order to adjust the ASR's workload in LAWS's mechanism. By using our scheme, we achieve more than 1.5x speedup compared to UNFOLD in all the LAWS configurations. The main reason of this improvement is the decrease of ASR's workload, which is resulted by combining the hypotheses' data-locality with their likelihood in driving the Viterbi search. With the conservative selection of LAWS beam width of 9, we can reduce the decoding time by 1.82x, whereas by aggressively pruning hypotheses with bad temporal locality, LAWS largely reduces workload with small impact on accuracy. Regarding the policy of having single beam, LAWS reaches to 2.7x speedup by using the most aggressive configuration. However, by having only one beam, LAWS suffers some accuracy loss near to 1% (see Figure 5.14). This is because that LAWS discards the hypotheses requiring off-chip accesses at all the frames similarly. On the other hand, by applying two different beams at the same time and selecting them based on different confidence thresholds, we can adjust the workload reduction in various speech frames and achieve high speedup with better accuracy. The more aggressive we set the threshold on the number of hypotheses, the higher the performance improvement and WER. For instance, considering *2avg* as the threshold, we obtain 2.2x speedup with a negligible increase of 0.2% in WER (see Figure 5.17).

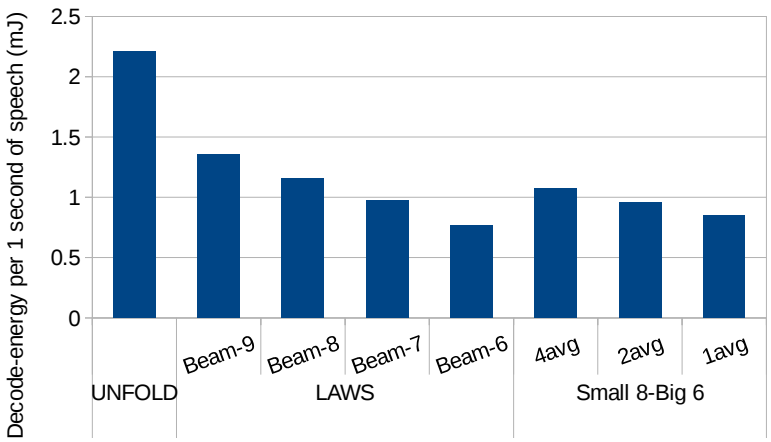


Figure 5.25: ASR's energy-consumption for decoding one second of speech for the baseline and different configurations of LAWS using single and dual beams.

Another main benefit of LAWS, consequent to the ASR's workload reduction, is a significant saving in energy consumption. Figure 5.25 shows the energy consumed for decoding one second of speech in the baseline and different LAWS's configurations. As seen, with respect to UNFOLD, our scheme saves energy by more than 1.91x in all the configurations. Similar to the performance improvements, $Small_8$-Big_6 provides better trade-off in accuracy and energy reduction. Regarding the threshold of *2avg*, the energy-consumption decreases by 2.3x. Moreover, we can gain 30% higher energy reduction using a more aggressive threshold as *1avg*. However, doing so can increase the WER by 0.32%, which may be acceptable for most applications. Thus, by reducing the decoding hypotheses through exploiting data-locality, combined with beam pruning, we achieve significant improvements in energy consumption and performance of ASR systems. Furthermore, we maintain the ASR accuracy by adapting its workload reduction using the search confidence throughout the Viterbi's run-time.

In addition to the benefits in energy reduction and performance, LAWS solves another main challenge of ASR systems, by reducing the memory requirements by almost an order of magnitude. Figure 5.26 illustrates the normalized memory requests and bandwidth required for UNFOLD and LAWS using the different configurations. As depicted, by using single beam, LAWS reduces memory activity to less than 17%. In addition, the dual beam approach shows between 88% to 90% decrease in memory usage considering the different confidence thresholds. We get this huge reduction in memory requirements since LAWS removes most of the memory fetches required for the hypotheses whose data is off-chip. Regarding the bandwidth, we save more than 70% in all the configurations. By reducing the beam, bandwidth requirement gets as low as 356 MB/s, reduced by almost 75% compared to the baseline. Furthermore, $Small_8$-Big_6 achieves between 73% to 74.5% saving in bandwidth. As LAWS performs much faster than UNFOLD, the results of bandwidth savings show lower numbers than the reduction in memory requests.

Finally, we have evaluated the area usage and power dissipation of our design, comparing the overhead added to the UNFOLD's architecture. Our synthesis results show an area increase of 7.5% in the State Issuer to implement the LAWS's mechanism and less than 1% in the State Cache, which results in less than 0.1% increase in Viterbi's area. The power is almost similar for most of the

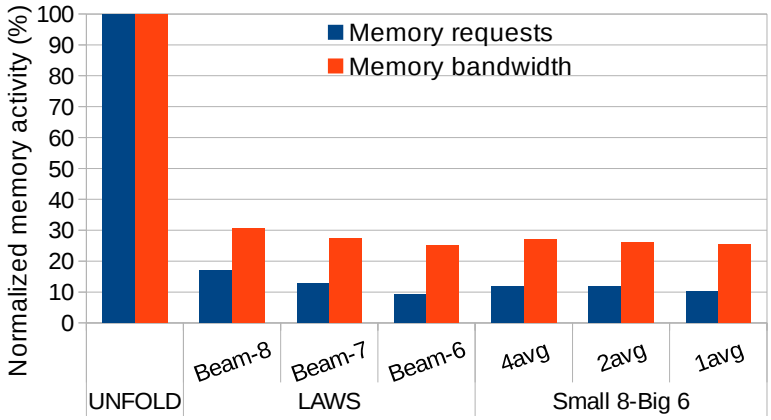


Figure 5.26: Normalized percentage of memory requests and bandwidth for the baseline accelerator and the different configurations of LAWS. The baseline memory bandwidth is 1.43 GB/s.

accelerator's components, except for main memory, which has been reduced by 73%. The reason is that we improve performance and at the same time we save energy by a large amount by removing most of the ineffective off-chip hypotheses. Overall, we get a power reduction between 5.2% and 5.9%.

6

Fully-Accelerated ASR Design

In this chapter, we present an end-to-end acceleration solution for the hybrid ASR model, including UNFOLD accelerator for the Viterbi beam search and a DNN ASIC to implement acoustic scoring on hardware. We have shown in the previous chapter that UNFOLD performs the Viterbi search 155x faster than real-time and it is no longer the main bottleneck of the ASR's pipeline. Regarding the DNN accelerator, we leverage the state-of-the-art solutions and evaluate how they behave in the context of ASR when considering the entire pipeline and the interaction among different components. More specifically, we analyze widely used DNN optimization techniques, such as aggressive quantization or DNN pruning, and evaluate their impact on the performance and energy consumption of the entire ASR system. We show that these DNN optimizations are effective when considering the isolated DNN accelerator, as the DNN is still able to identify the top-1 phoneme on each frame, but the distribution of likelihoods is significantly affected and the confidence of the prediction is largely reduced. Since Viterbi is the consumer of DNN-computed likelihoods, reduction in DNN confidence produces an explosion in the number of hypotheses explored, resulting in large slowdowns in the Viterbi accelerator. To solve ASR workload's increase, we propose to modify the Viterbi accelerator in order to limit the search expansion to the Best-N hypotheses at each speech frame. By doing so, we effectively integrate DNN optimizations with the Viterbi accelerator, achieving large reductions in execution time and energy consumption for the overall ASR pipeline.

6.1 DNN optimizations for Kaldi's ASR Network

In this section, we analyze the impact of DNN pruning and weight/input linear quantization 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 or quantized models,

Table 6.1: Kaldi's DNN structure 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.

Layer	Neurons	Weights
FC0	360	129K
FC1	2000	720K
P1	400	0
N1	400	0
FC2	2000	800k
P2	400	0
N2	400	0
FC3	2000	800k
P3	400	0
N3	400	0
FC4	2000	800k
P4	400	0
N4	400	0
FC5	3482	1.4M
SoftMax	3482	0

considering not only the top-1 or top-5 error, but also the confidence and reliability 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.

6.1.1 Kaldi's DNN

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 6.1. 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) [38]. Next, the DNN consists of 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 6.1 also shows the number of neurons and weights in each layer. Kaldi's DNN contains more than 4.5 million parameters.

Table 6.2: DNN pruning at the three ascending global rates of 70%, 80% and 90%. The percentages of pruning are reported based each layer of the Kaldi's DNN. We only show the FC layers as they are the ones with parameters to be pruned away.

Layer	70%-Pruning	80%-Pruning	90%-Pruning
FC0	0	0	0
FC1	71%	82%	92%
FC2	68%	80%	92%
FC3	65%	77%	91%
FC4	95%	98%	99%
FC5	66%	78%	90%

6.1.2 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 [29]. 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 [114, 39, 42] 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. [42] 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 [42]. 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.

In order to achieve a global pruning of 70%, 80% and 90%, we set the quality parameter, as specified in [42], to 1.44, 1.90 and 2.71 respectively. Table 6.2 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 [38]. Nevertheless, its weights are accounted for the total model size. The original non-pruned DNN is trained using the training dataset from LibriSpeech [72]. This training dataset is also used for the retraining after the pruning.

6.1.3 Weight and Input Quantization

DNN quantization is a technique used for reducing both computational cost and memory bandwidth usage, resulting in lower energy-consumption for DNN inference. However, by applying quantization we may encounter some accuracy loss that can be controlled by carefully assigning

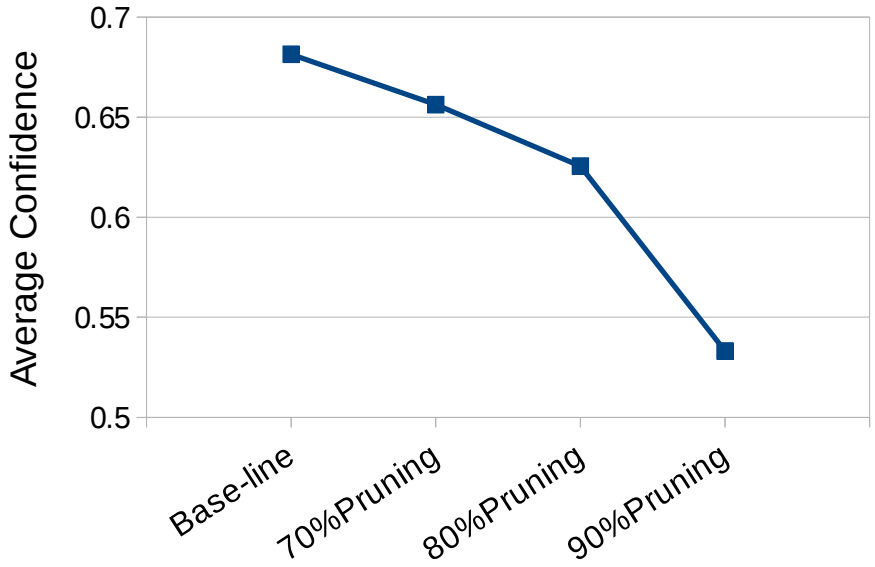


Figure 6.1: Average DNN confidence for non-pruned DNN and three pruned models with 70%, 80% and 90% pruning respectively. DNN confidence is significantly decreased when applying pruning.

the required amount of precision to each DNN layer. To this end, we analyzed the performance-accuracy trade-off for the different fully-connected layers of Kaldi's DNN, when reducing the number of bits employed to encode inputs and weights. Under linear quantization, data (inputs/weights) are represented by an integer index on a specific range. By identifying the range of the neuron's inputs and weights, we uniformly distribute various number of clusters based on the number of bits decided for representing each value. We do this analysis for both the DNN weights and a large number of input datasets used at the training phase.

In this chapter, we try to reduce the precision of the weights and inputs as much as possible without affecting Word-Error-Rate (WER). We found that by reducing the precision from 32-bit floating-point to 16-bit integer operations, the accuracy change is very negligible (less than 0.01%). Thus, we use this configuration as our baseline. The number of bits for inputs and weights may be further reduced with a small impact on accuracy, but at the cost of significantly increasing the workload of the Viterbi search as we show in Section 6.1.6. More specifically, regarding the weight quantization, we can only reduce the number of clusters to 128, otherwise we experience sudden increase in the WER (from 10.5% to 95%). On the other hand, we can decrease the input clusters to as low as 32, which corresponds to 5-bit data representation, without any serious accuracy loss.

6.1.4 Confidence of the Pruned DNN

We analyze in this section the impact of pruning in the state-of-the-art Kaldi's DNN used for speech recognition. 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

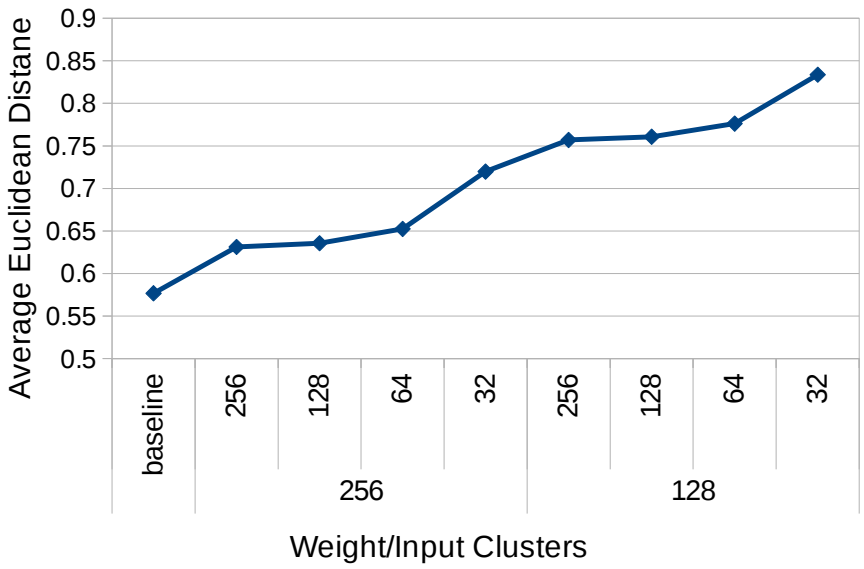


Figure 6.2: Average euclidean distance for the baseline and the different quantized version of Kaldi's DNN. The horizontal and vertical numbers on the x-axis show the weight and input clusters, respectively. By reducing the precision, euclidean distance increases substantially, showing the low-reliable DNN classification.

likelihood, i.e. the top-1 class. Figure 6.1 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 a slowdown of 4.5x in the Viterbi beam search, and an overall slowdown of 33% in the ASR system.

6.1.5 Increase of Euclidean Distance

As previously mentioned in Chapter 1 and Chapter 2, DNN evaluation is a key component of the ASR system, as it feeds the Viterbi search with the output probabilities of the different audio phonemes in order to decode the speech input. The DNN's performance and accuracy is crucial for the Viterbi search expansion, which defines the complexity of the search and its execution time. Therefore, the DNN is not an isolated part and the rest of the ASR pipeline depends on how accurately it classifies the correct phoneme from each frame's feature-vector. Moreover, Viterbi search does not use only the highest output probability (top-1), but it relies on the whole probability spectrum of the phonemes to produce multiplicative number of hypotheses and then, after processing all the speech frames, backtrack the most likely hypothesis in order to decode the utterance. We introduce euclidean-distance as a metric to assess the quality of DNN-computed

likelihoods compared to the ideal classification.

Both DNN pruning and quantization have the side effect of increasing the euclidean distance of the evaluated DNN compared to the one-hot classification. We define the one-hot classification as distribution of likelihoods where the correct class has the highest probability (1.0), while assigning the rest of output classes at the lowest possible likelihood (0.0). Then, we calculate euclidean-distance as follows:

$$Euclidean\text{-}Distance = \sqrt{\sum (out_{\text{one-hot}}(j) - out_{\text{eval}}(j))^2} \tag{6.1}$$

where out_{eval} is the current evaluated output of the DNN, whereas $out_{\text{one-hot}}$ is the synthetic output for the one-hot classification. As this equation shows, the summation of the difference between the DNNs' outputs should get near zero if the evaluated network classifies with high accuracy and confidence. However, we have seen that either DNN pruning or weight/input quantization causes the sudden increase in the euclidean-distance for the DNN evaluation.

Figure 6.2 shows the change in the euclidean-distance of the Kaldi's DNN using both weight and input quantization with integer operations, compared to the baseline which uses 32-bit floating-point precision. As illustrated, even by reducing the precision to 8-bit, which is common for most of DNN acceleration designs, the average euclidean-distance increases by 10%. As ASR accuracy remains almost unaffected (0.3% of WER increase), we suspect that the low-confident behaviour of the DNN classification is the main reason for such change in the euclidean distance. By further reducing input clusters, the euclidean-distance grows as high as 0.72 (25% of increase). On the other hand, euclidean-distance is more sensitive to the weight precision. By only removing 1 bit (half of the clusters) for the weight representation, we encounter 31% increase and up to 45% if the most aggressive input quantization is applied. Note that we did not extend this analysis for the lower number of clusters as ASR fails to decode the speech (more than 95% of WER).

6.1.6 Impact of DNN Pruning and Quantization 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 [106], which is claimed to achieve human parity in speech recognition, employs DNN-computed scores to drive a Viterbi search [62]. Recent works on hardware-accelerated ASR also integrates a DNN with a Viterbi search [80, 110, 111, 108]. On the other hand, solutions based on Recurrent Neural Networks (RNNs) also include a Viterbi beam search. For example, Baidu's DeepSpeech [43] combines an RNN with the Viterbi search algorithm as described in [44], whereas EESEN [63] employs a Viterbi search driven by the scores computed by an LSTM network [40, 92]. Furthermore, regarding the GMM-based solutions, recent software optimization [56] and hardware acceleration [99] largely reduce GMM computation time, making the Viterbi search the main bottleneck. 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 and weight/input quantization on the performance of the Viterbi beam search [83], 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

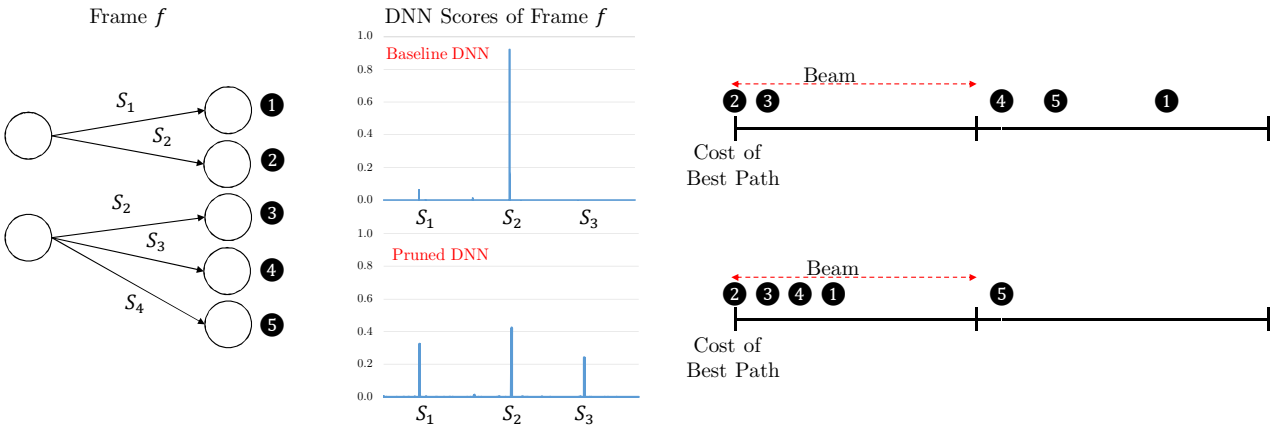


Figure 6.3: The figure illustrates the behavior of the Viterbi beam search for one frame of speech, for the non-pruned DNN with 32-bit floating-point precision (top) and the pruned/quantized model (bottom).

language. In a modern ASR system, the language is represented by using a Weighted Finite State Transducer (WFST) [66]. 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 [110, 108].

Figure 6.3 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 6.3 shows the costs of the different hypotheses when using the baseline DNN (non-pruned model with 32-bit floating-point precision). 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 [108]. 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 baseline DNN is very confident

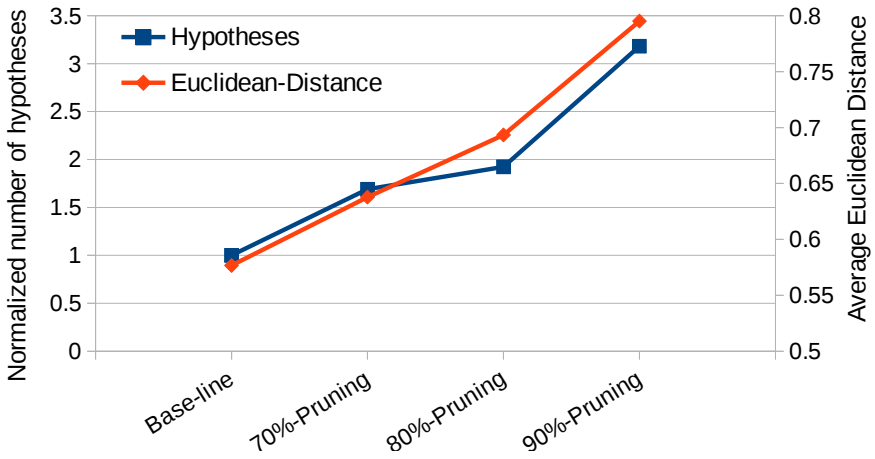


Figure 6.4: Normalized number of hypotheses, a.k.a. paths, explored during the Viterbi search and euclidean-distance for the pruned DNN models at 70%, 80% and 90% of pruning. DNN pruning has a high impact on the workload of the beam search.

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 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 6.3 shows the costs of the different paths when using the pruned/quantized model. This 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 baseline model, and fall within the beam distance. Therefore, 1 and 4 are not discarded when using the low-confident 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.

Figures 6.4 and 6.5 show the increase in Viterbi workload when using the pruned DNN models from Table 6.1 and the different quantized models, respectively. Moreover, we include the euclidean-distance in theses two figures in order to illustrate the close correction between the ASR workload with the confidence or reliability of the DNN outputs. As it can be seen, Viterbi workload increases when raising the degree of pruning or reducing the quantization precision, i.e. when the confidence of the DNN is reduced. Regarding the pruned models, although the 70% of pruning only introduces a confidence loss of 5% and 11% increase in the euclidean-distance, 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%, with 20% higher euclidean-distance, causing an increase of almost 2x in the Viterbi search workload. At the highest ratio, the 90% pruning introduces a confidence loss of 22% in DNN predictions, 38%-increased euclidean-distance and an increase of more than 3x in the number of hypotheses explored. On the other hand, with the quantized models, we

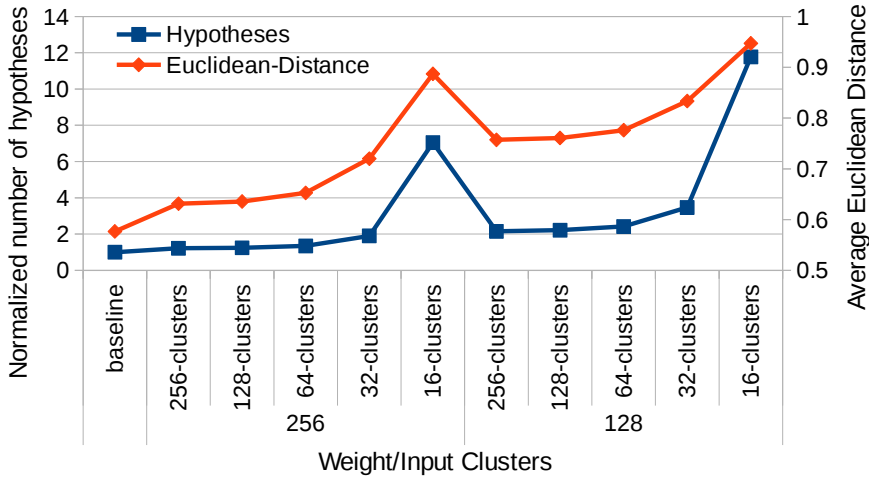


Figure 6.5: Normalized number of hypotheses, a.k.a. paths, explored during the Viterbi search and euclidean-distance for the different quantized DNN models. DNN quantization reduces the reliability of the DNN's outputs and, consequently, has a high impact on the workload of the beam search.

can see that the workload significanly increases when using more aggressive quantization, and the workload increase is well correlated with the increase in euclidean-distance. By a small decrease of the DNN's outputs reliability, Viterbi has to explore a significantly higher region than before in order to compensate for the lost confidence in the acoustic-scores and preserve the ASR's accuracy. This increase in activity results in the large slowdowns for the Viterbi search and the whole ASR application as previously reported in Figure 1.8.

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 and quantization 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.

6.2 Hardware-Accelerated ASR

In this section, we present our hardware-accelerated ASR system that manages to efficiently combine DNN pruning with Viterbi beam search. Furthermore, our system supports for the low-precision quantization of the DNN's weights and inputs, by effectively reducing the Viterbi workload. First, we describe the baseline Viterbi search accelerator, focusing on the components that are affected by the workload increase due to either the pruning or quantization (see Section 6.1.6). Next, we describe how the architecture of the Viterbi accelerator can be extended to mitigate the

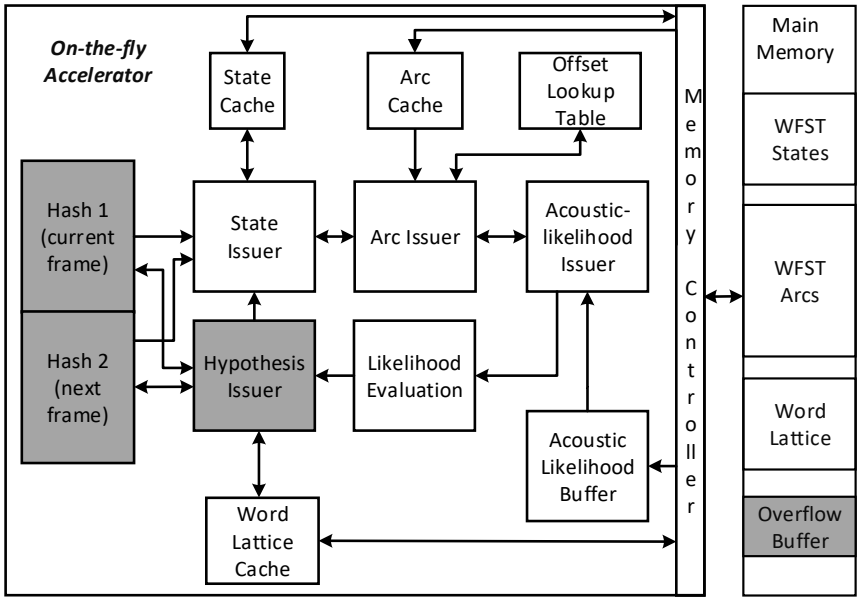


Figure 6.6: Architecture of UNFOLD [108], a state-of-the-art Viterbi search accelerator. The main components affected by DNN optimizations are highlighted.

negative effects of DNN optimizations, 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/quantized DNNs.

Our ASR system employs UNFOLD [108] to achieve high-performance and energy-efficient Viterbi search. Figure 6.6 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 optimizations, which are later modified in Section 6.2.1 to mitigate the negative effects of the Viterbi search explosion. These components are highlighted in Figure 6.6. A more thorough description of UNFOLD is provided in Chapter 5.

6.2.1 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

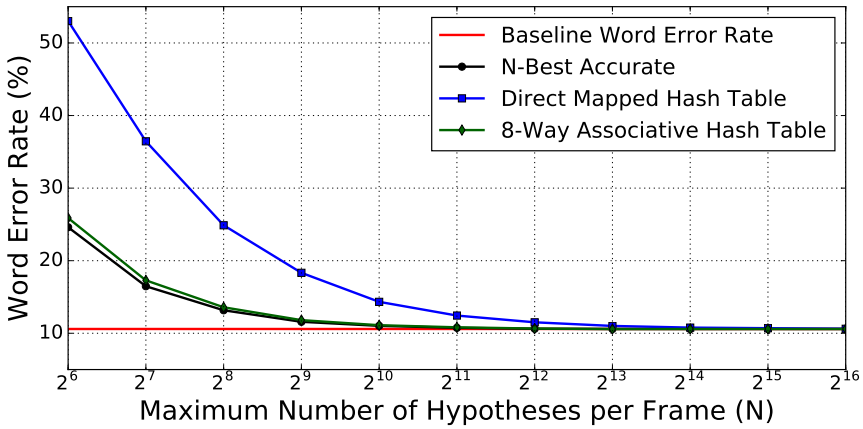


Figure 6.7: Word Error Rate (WER) versus maximum number of hypotheses explored per frame (N). An 8-way associative hash table with 1024 entries (128 sets) achieves nearly the same WER than the baseline system that explores an unbounded number of hypotheses. In addition, our 8-way associative hash table exhibits very similar behavior to the system that accurately tracks the N-best hypotheses (N-Best Accurate).

correspond to the same entry are linked. Therefore, accessing 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 [72] test set audio files, and found that the average number of hypotheses per frame is larger than 20K, whereas the maximum number gets bigger than 300K. UNFOLD's hash table [108] 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, non-quantized 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/quantized DNN, the number of hypotheses increases significantly as shown in Figures 6.4 and 6.5, 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 a 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/quantized DNNs.

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

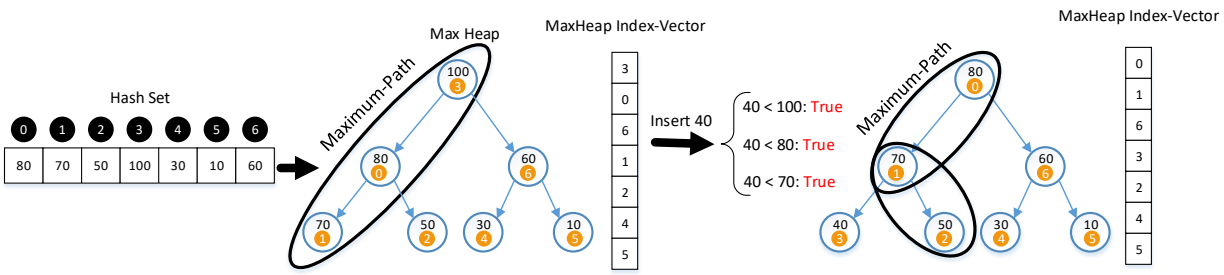


Figure 6.8: An example of the Max-Heap binary tree for a set with 7 hypotheses, that illustrates how the replacement of the worst hypothesis is performed in our hash table.

accesses to the overflow buffer in main memory. Note that states in the WFST have multiple outgoing arcs (see left part of Figure 6.3) 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 6.7 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 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 associative 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 or quantization, 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 6.2.1).

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 6.8 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 6.8, 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 6.8. 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 6.8 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).

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

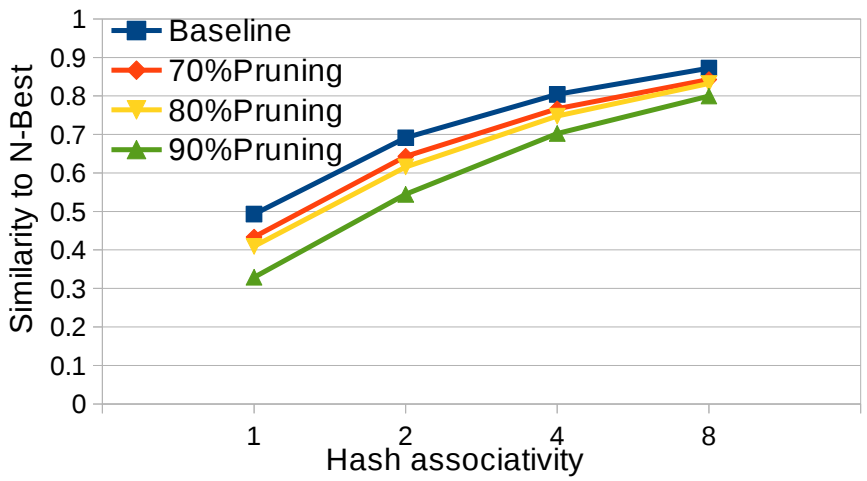


Figure 6.9: Similarity between a system that accurately selects the N best hypotheses and our system that loosely tracks the N best paths, for different degrees of pruning and associativities.

overhead of 6% and 0.2% in the total area and power dissipation respectively.

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 6.9 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. Furthermore, we have seen very similar behaviour when using the low-precision quantization technique for the DNN's weights and inputs.

6.2.2 DNN Accelerator Overview

In this subsection, we describe the DNN accelerator employed in our ASR system. This DNN accelerator is loosely based on DaDianNao [20], but we have extended it to support pruned models and different quantization precision. A high-level block diagram of the accelerator is provided in Figure 6.10. 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

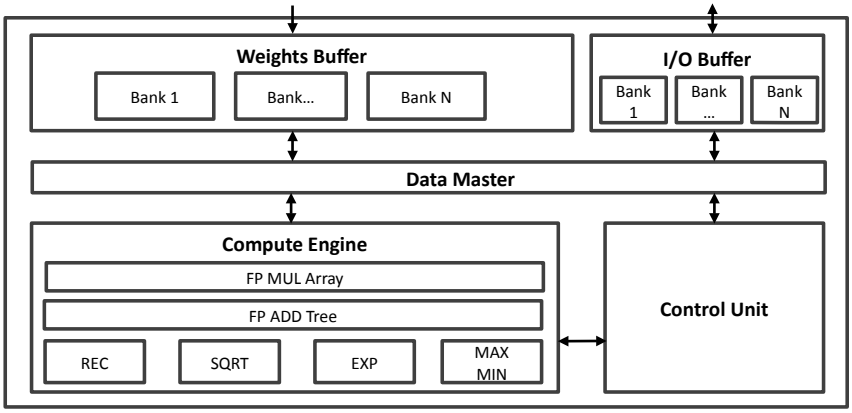


Figure 6.10: Architecture of the DNN accelerator used in our ASR system.

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 6.3. 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 6.3. 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.

Furthermore, we have changed the CE when processing weights and inputs with low-precision quantization, in order to reduce both the complexity of multiply-and-adder units and the pressure on I/O and weight buffers for fetching the data. Doing so, we save the energy consumption of the accelerator by 1.82x.

Multiple instances of the accelerator shown in Figure 6.10 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.

6.3 Experimental Results

This section presents the experimental results achieved by our ASR system that efficiently combines the DNN pruning and quantization with the Viterbi beam search. The baseline configuration is a hardware-based ASR system that includes two accelerators: the DNN accelerator optimized for pruned/quantized fully-connected networks described in Section 6.2.2, and the state-of-the-art Viterbi search accelerator presented in Section 5.1. The configuration parameters used in each accelerator are shown in Table 6.3 and Table 6.4. For the Viterbi accelerator, most of the parameters are taken from [108], 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 6.3: 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 6.4: 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

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. Furthermore, we use a modified version for the DNN accelerator, supporting the low-precision quantization for either DNN's weight of inputs. We name these configurations as *Baseline-X-Y*, where X and Y show the number of bits used for weights and inputs, respectively.

In order to deal with the increase in Viterbi search workload due to the DNN pruning (see Section 6.1.6), 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 5.1. We use 1024 as the value of N. We label these configurations as *NBest-NP* (non-pruned), *NBest-70*, *NBest-80* and *NBest-90*. Regarding the DNN quantization scheme, we use similar terminology except for using the X and Y instead of the pruning ratio (*NBest-X-Y*), showing the quantization bit-width for weights and inputs, respectively.

Figure 6.11 and 6.12 show the normalized execution time for the different configurations considering DNN pruning and quantization, respectively, 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 or *Baseline-FP*, i.e. the baseline hardware-accelerated system with the non-pruned and non-quantized (full-precision) 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. On the other hand, by using low-precision quantization technique,

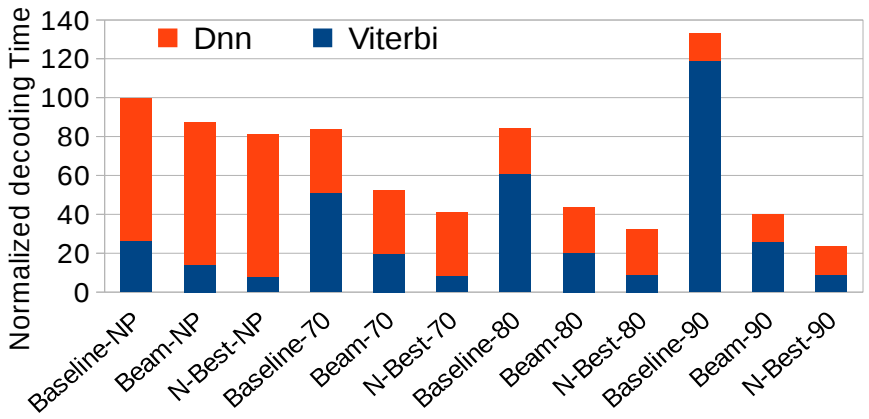


Figure 6.11: Execution time for the entire ASR system, including the breakdown between execution time of DNN and Viterbi accelerators. Time is normalized to the configuration Baseline-NP, i.e. the baseline ASR system with the non-pruned DNN.

we save the energy of DNN by more than 2x. Note that *Baseline*,*Beam* and *NBest* employ the same DNN accelerator, they only differ in the implementation of the beam search. Both pruning and quantization introduce large slowdowns for Viterbi in the *Baseline* ASR system, producing an increase in execution time of the beam search by 1.9x, 2.3x and 4.5x for 70%, 80% and 90% of pruning, and from 1.3x to 3.2x for the weight and input quantization. The performance loss in Viterbi search offsets part of the benefits from DNN pruning for *Baseline-70* and *Baseline-80*, and it produces slowdown for the entire ASR system for *Baseline-90* (33%) and also in different quatized versions of DNN (5%-58%). Even though DNN pruning, except for the most aggressive case of 90%, still shows a viable technique to reduce the energy and time of acoustic-scoring, however it damages the Vitebi decoding process in ASR which can produce the long tail-latency for the noisy and hard-to-decode utterances, resulting in poorer performance than the baseline. This slowdown is caused by the workload increase due to the reduction in DNN confidence, as explained in Section 6.1.6, Figures 6.4 and 6.5.

Reducing the beam width in the *Beam* configurations of Figure 6.11 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. Due to such inefficiencies, we only show the result for our scheme regarding the low-precision quantization in Figure 6.12.

As an alternative to the *Beam* configuration, 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, non-quantized

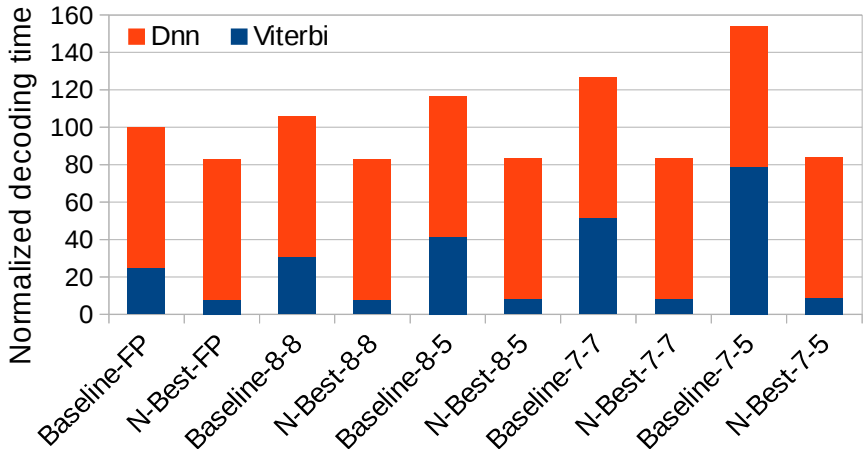


Figure 6.12: Execution time for the entire ASR system, including the breakdown between execution time of DNN and Viterbi accelerators. Time is normalized to the configuration Baseline-FP, i.e. the baseline ASR system with the full-precision (Floating-Point) operations.

DNN, *NBest-NP* (*NBest-FP*) 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 either pruning or quantization. *NBest-90* achieves 4.2x speedup with respect to *Baseline-NP*. In addition, it improves the performance of the *Beam-90* by 1.69x. On the other hand, we achieve almost similar 1.22x speedup for all the quantized versions of DNN, as we are not changing its computation workload and also the number of resources for DNN accelerator, and the improvement is mainly because of Viterbi search optimization which also prevents the explosion in the search expansion. Therefore, our *NBest* solution is able to effectively combine both DNN pruning and quantization with the beam search, providing an implementation where the complexity of the search is not affected by the reliability issue, i.e. euclidean-distance increase of the DNN predictions.

Figure 6.13 and 6.14 show the normalized energy for the same configurations, including both static and dynamic energy. Regarding the DNN pruning, it 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. On the other hand, we obtain between 1.88x to 1.91x energy-saving by using low-precision quantization. The reason for lower energy-reduction of quantization technique compared to pruning is that the DNN computation remains the same as well as the memory-accesses to the on-chip buffers. However, the functional units become less complex and the required bandwidth drops, which results more than 1.82x energy reduction for DNN.

Regarding the Viterbi accelerator, pruning and quantization increases its energy consumption in the *Baseline* system by up to 4.3x and 4.1x, respectively. This increase in energy consumption is due to the large workload increase caused by the increase in euclidean-distance of DNN outputs. The Viterbi accelerator explores more paths when using the pruned/quantized models, as previously reported in Figure 6.4, which requires more memory accesses and computations. The *Beam-90* configuration in Figure 6.13 suffers an increase in Viterbi search energy consumption of

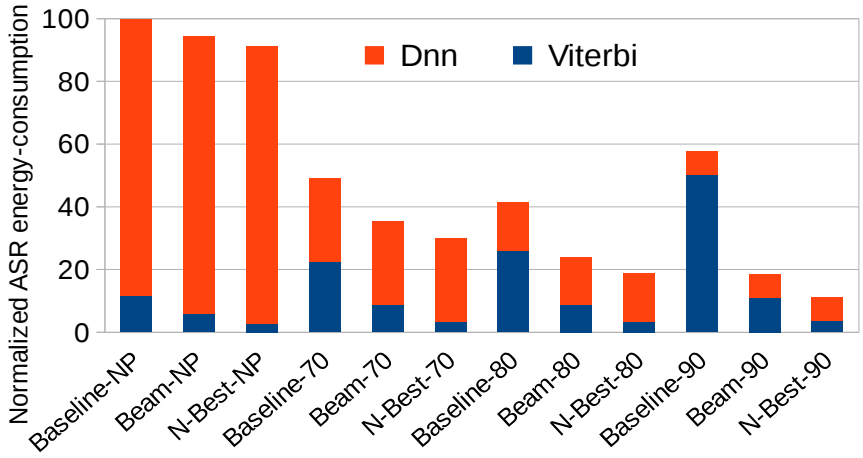


Figure 6.13: Normalized energy for the entire ASR system, including the breakdown between energy consumption of the DNN and Viterbi accelerators. Baseline configuration is labeled as Baseline-NP, i.e. the baseline ASR system with the non-pruned DNN. Numbers include both static and dynamic energy.

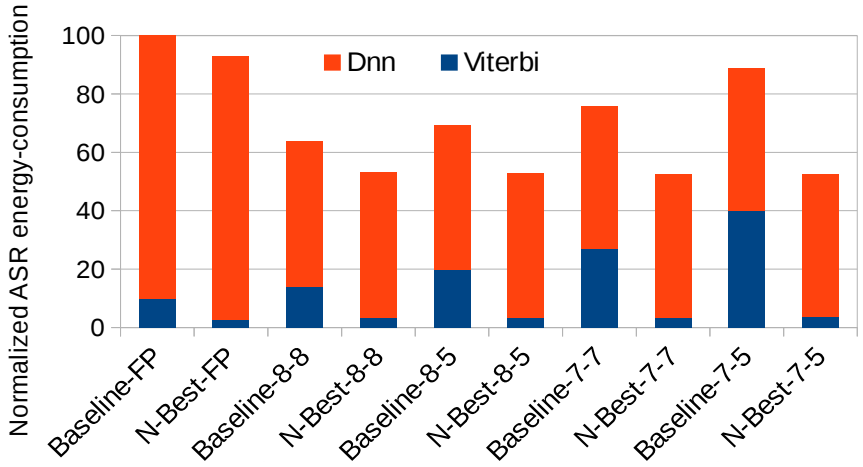


Figure 6.14: Normalized energy for the entire ASR system, including the breakdown between energy consumption of the DNN and Viterbi accelerators. Baseline configuration is labeled as Baseline-FP, i.e. the baseline ASR system with the full-precision (Floating-Point) operations. Numbers include both static and dynamic energy.

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. In addition, our approach solves the negative impact on Viterbi search regarding the quantization scheme (see Figure 6.14), resulting in 2.7x and 1.91x energy-saving for the Viterbi search and the whole ASR, respectively.

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 5.1. 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 Figures 6.11 and 6.12. In addition, the area of our accelerator is reduced since the hash table employed to maintain the hypotheses is

smaller (see Table 6.4). 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 [108] 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 addition to evaluating the pruning and quantization effects separately, we have applied both techniques in order to measure their combined impact on either the ASR accuracy and also the Viterbi workload increase. For instance, we have experimented the various numbers of quantization clusters on to of the 70%-pruned DNN model. The WER numbers show that by reducing the weight and input precision from 16 to 8 and 7, respectively, we can increase the WER from 10.48% to 12.64%. By applying more aggressive quantization to either weight and input, we incur significant drop in accuracy (higher than 10% increase in the WER). Moreover, just by changing the number of quantization bit from 16 to 8 for both the input and weight, we get 3.1x increase of the number of the hypotheses evaluated by the Viterbi search. By using our technique, *NBest*, we handle the Viterbi's workload the same way as illustrated by Figures 6.4 and 6.5. Finally, we can achieve an speedup of 2.4x and an energy-saving of 6.3x compared to the baseline DNN, when combining 70% pruning with the low-precision quantization of 8-bit for the input and weight.

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. Furthermore, we can get significant energy reduction by using weight and input quantization. Considering DNN pruning, 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.

7

Conclusions and Future Work

In this chapter, the main conclusions and contributions of this thesis are presented, as well as some open-research areas for future work.

7.1 Conclusions

The purpose of this thesis is to design a customized architecture for ASR systems, as one of the popular cognitive computing applications, in order to not only accelerate the performance, but also bring significant efficiency in power, area and memory usage. In this regard, we have implemented several versions of an ASR accelerator considering different trade-offs for the different architectural metrics.

As the initial phase, we characterize the whole ASR's pipeline on both CPU and GPU. We observe that out of the two main stages, i.e. DNN and Viterbi search, Viterbi is by far the main bottleneck of the system consuming more than 73% and 85% in CPU and GPU, respectively. We then design a hardware accelerator specialized for this specific task. By our first design, we could achieve the same performance as high-end desktop GPU (GTX980), while reducing the energy-consumption by two orders of magnitude. Furthermore, we analyze the main performance bottleneck for the accelerator, detecting the memory subsystem as the main source of pipeline stalls. In order to ameliorate the memory bottleneck, we propose a new prefetching scheme which is based on the decoupled access-execute architectures and computes the next memory requests while processing the Viterbi algorithm on the speech graph. By doing so, we achieve around 2x speedup. Furthermore, we propose a bandwidth-saving technique that removes most of the indirections required for locating the arcs of a given state in the WFST.

By the first design of the Viterbi accelerator, we could optimize the ASR system in both perfor-

mance and energy budget with a relatively small area usage of nearly 24 mm^2. In order to perform ASR, there is a huge memory requirement of normally above 1 Gigabyte for the speech dataset and a peak memory bandwidth usage of more than 10 GB/s [109, 108]. These requirements become an important bottleneck when applying ASR on small-form mobile systems, wearables and IoT devices. Thus, we propose to change the way Viterbi algorithm is handled in the hardware by taking the on-the-fly composition approach on the large speech graph. Under on-the-fly composition, we firstly divide the graph's structure into significantly smaller models, i.e. Language Model (LM) and Acoustic Model (AM), by reverse-engineering the training process of building the fully-composed graph. As a result, we get a 10x reduction for the speech dataset: size of the WFST is reduced from more than 1 GB to almost 100 MB. Furthermore, we utilize some hardware-friendly yet effective compression techniques on both LM and AM. We exploit individual data properties of LM and AM in order to shrink the memory footprint to less than 40 MB, yielding 31x compression ratio on average for the different ASR decoders.

In order to employ the on-the-fly composition in the Viterbi search accelerator, we make some relevant changes to our first design, resulting in a memory- and power-efficient architecture called UNFOLD. UNFOLD implements the Viterbi search differently, by traversing AM and LM at the same time and composing them on-the-fly whenever necessary in the search expansion. Thus, this algorithm is taking less memory space, as it is not using the enormous offline-composed speech graph for decoding the input utterance, whereas it is constructing the required part of the dataset based on the input data. UNFOLD reduces the power dissipation of the Viterbi search by 2x, and saves the total energy by around 30%. Moreover, it uses 2x less area than our previous design, since it requires smaller caches and on-chip buffers due to the decrease in the memory footprint due to on-the-fly composition and compression techniques. Finally, UNFOLD performs 155x faster than real-time, with a negligible slowdown of 18% on average compared to our previous design, as it performs both the Viterbi search and composition at the same time, imposing higher workload.

In spite of UNFOLD's benefits in providing a more efficient memory subsystem, ASR still requires 1.5 GB/s memory-bandwidth that can be inapplicable for wearables and IoT devices. Moreover, the off-chip memory requests are the most expensive operations considering the whole accelerator's activity, as they require three orders of magnitude higher energy consumption than the rest of the computations or on-chip memory accesses [39]. Therefore, as an extension for the UNFOLD architecture, we propose a Locality-AWare Scheme (LAWS), which reduces the ASR's memory bandwidth usage by discarding unlikely hypotheses whose data is not in the on-chip memories. Unlike conventional schemes to prune the search space, our technique combines both the likelihood of the hypotheses and their locality. In addition, our technique dynamically adapts the beam based on the confidence of the decoder in selecting the correct path for the speech decoding, significantly reducing ASR workload. LAWS provides 2.2x speedup and 2.3x energy-saving with negligible accuracy loss.

After optimizing the Viterbi search, we concentrate on the other stage of ASR pipeline, i.e. the DNN-based acoustic-scoring, as it becomes the main bottleneck of the system. We leverage the state-of-the-art DNN acceleration design in order to improve its performance. Furthermore, we apply several DNN optimization techniques such as DNN pruning and low-precision quantization. Even though the acoustic-scoring accuracy does not change by either pruning or quantizing the DNN, we have seen that the probability of top-1 audio phoneme significantly reduces, introducing

less confident and reliable predictions. By using the pruned/quantized acoustic-scores, Viterbi search suffers large slowdown due to the increase of the search expansion in order to maintain the ASR accuracy. For instance, when using a DNN pruning degree of 90% we encounter 33% of performance slowdown in the ASR pipeline due to the workload explosion of the Viterbi search. In order to deal with this problem, we propose to change the Viterbi search accelerator to restrict the number of search-paths to the N-Best hypotheses at each frame of speech. Our solution achieves a good trade-off between accuracy and performance, even by using a small number of paths (N) such as 1K. Regarding the hardware implementation, we use the set-associative hash structure in order to loosely track the N-Best hypotheses, since accurately selecting the N-Best would require an expensive sorting. Our end design is able to effectively combine DNN optimization techniques with the Viterbi beam search, resulting in 4.2x performance speedup and 9x energy reduction.

7.2 Contributions

In this thesis, we evaluate the acceleration design for a complete ASR pipeline, using the state-of-the-art approach for speech recognition systems. Doing so, we have explored the different challenges regarding the performance, power and memory requirements of various ASR decoders. As the result of this thesis, we provide several accelerator's designs considering the different trade-offs of the architecture. The main contributions obtained through this dissertation are summarized as follows.

In first place, we design a custom hardware accelerator for large-vocabulary, speaker-independent, continuous speech recognition, motivated by the increasingly important role of automatic speech recognition systems in mobile devices. Our design includes innovative techniques to deal with memory accesses, which is the main bottleneck for performance and power in theses systems. The final design including both improvements achieves a 1.7x speedup with respect to a modern high-end GPU, while reducing energy consumption by 287x. This work has been published in the 49th international symposium on Microarchitecture (MICRO):

- ✓ Reza Yazdani, Albert Segura, Jose-Maria Arnau, Antonio González, "An Ultra Low-Power Hardware Accelerator for Automatic Speech Recognition," 49th Annual IEEE/ACM International Symposium on Microarchitecture (**MICRO)**, Taipei, Taiwan, 2016.

As a complete evaluation for the ASR pipeline, we propose an SoC system architecture combining the Viterbi accelerator with Tegra-X1 mobile SoC that contains an ARM CPU and the Nvidia GM204 GPU. Our system outperforms traditional solutions running on the CPU by orders of magnitude. Compared to a GPU-only system, out hybrid scheme improves performance by 5.25x while reducing energy by 2.05x. This work has been published in IEEEMicro, 2017:

- ✓ Reza Yazdani, Albert Segura, Jose-Maria Arnau, Antonio González, "Low-Power Automatic Speech Recognition Through a Mobile GPU and a Viterbi Accelerator," IEEE Micro, 2017.

To further decrease the power dissipation, we propose the drowsy technique for the hash microarchitecture of the Viterbi accelerator. This scheme dynamically decides on the state of each

memory partition that whether it needs to be active or drowsy based on the access pattern of fetching its data from the ASR pipeline. It reaches to an almost 30% power reduction in the best case and close to 15% on average considering different ASR decoders. This work has been submitted to the journal of IEEE Transactions on Computers:

◇ Reza Yazdani, Jose-Maria Arnau, Antonio González, "A Low-Power, High-Performance Speech Recognition Accelerator".

In the next step, we address the main problem of previous accelerators for ASR: the large size of the WFST, i.e. the graph-based language database, used in the Viterbi search. We identify that its excessive size is due to the offline full-composition of the Acoustic Model (AM) and the Language Model (LM) during training stage, and propose to dynamically compose both models on demand at the decoding stage. UNFOLD operates on the smaller AM and LM models, providing simple yet efficient hardware for on-the-fly WFST composition. Therefore, our system does not require a large WFST composed offline, as it is able to dynamically obtain the parts required for decoding the input speech signal, reducing memory storage requirements by a large extent. Moreover, we introduce several compression techniques to further reduce the size of the AM and LM and a novel search pruning scheme, that the UNFOLD's architecture is extended to support. This work has been published in the 50th international symposium on Microarchitecture (MICRO):

✓ Reza Yazdani, Jose-Maria Arnau, Antonio González, "UNFOLD: A Memory-Efficient Speech Recognizer Using On-The-Fly WFST Composition," 50th Annual IEEE/ACM International Symposium on Microarchitecture (**MICRO)**, Los Angeles, CA, USA, 2017.

As an extension to UNFOLD, we target one of the main challenges for ASR systems at mobile, IoT and wearable devices, which is the high memory and energy requirements to perform speech recognition. Previous schemes try to solve this issue by discarding the unlikely hypotheses expanded by the Viterbi search using a beam pruning. Although reducing the beam width decreases ASR's workload significantly, it causes an important loss of accuracy. We present LAWS, a scheme that obtains high benefits in workload reduction without compromising accuracy by a new approach that combines new insights about data-locality characteristics of ASR with the statistical score during the Viterbi search. LAWS removes over 87% of the off-chip memory activity. This work has been submitted to the 28th international conference on parallel architectures and compilation techniques (PACT):

◇ Reza Yazdani, Jose-Maria Arnau, Antonio González, "LAWS: Locality-AWare Scheme for Automatic Speech Recognition".

As the last contribution, we analyze the impact of DNN pruning in the performance and energy consumption of an Automatic Speech Recognition (ASR) system. The confidence loss in the DNN outputs 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. In order to mitigate the negative effects of DNN pruning, we propose to extend the hardware of UNFOLD 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. This work has been published in the 45th international symposium on Computer Architecture (ISCA):

- ✓ Reza Yazdani, Marc Riera, Jose-Maria Arnau, Antonio González, "The Dark Side of DNN Pruning," 45th Annual IEEE/ACM International Symposium on Computer Architecture (**ISCA)**, Cambridge, MA, USA, 2018.

We have also extended this work in order to study the impact of the low-precision quantization on the ASR systems, and whether it affects the Viterbi's workload the same as DNN pruning. As mentioned in Chapter 6, the quantization increases the decoding time of ASR as well as DNN pruning by revealing some impact on the euclidean-distance of the DNN's outputs with respect to the correct classification. To overcome this issue, we take the same approach of selecting the N-Best hypotheses on each frame of speech, in order to keep the search space tractable and achieve the performance and energy improvements expected by the DNN quantization. This work has been submitted to the journal of IEEE Transactions on Computers:

- ◇ Reza Yazdani, Jose-Maria Arnau, Antonio González, "Negative Effects of Quantization and DNN Pruning on Automatic Speech Recognition".

7.3 Open-Research Areas

One extension of the work proposed in this thesis would be to improve the accelerated ASR design in a way that can easily adapt to the new modern and more complex speech recognition systems. The state-of-the-art approach for ASR decoders combines the DNN acoustic-scoring with Viterbi search on the speech graph and uses an RNN model as a rescoring phase, which significantly improves the accuracy. This improvement is due to the advances in DNN architectures for acoustic scoring [74] that provide higher accuracy for detecting the audio phonemes at each speech frame, and also due to the use of an RNN that works as an unbounded language model that stores all the word histories from the beginning of an utterance. Therefore, our system should be extended to accommodate these new DNN and RNN architectures that impose even higher computation and memory requirements.

RNNs represent a deep-learning model with increasing popularity for applications that are based on sequence-to-sequence processing [96], such as ASR decoders [63, 6]. A key feature of this type of neural networks is their use of past information for improving the accuracy. Long-Short-Term-Memory (LSTM) [45] is the most commonly used RNN. It can potentially remember useful information for a long period of time, providing high accuracy. However, RNN models are hard to accelerate as they come with too many recurrent steps and therefore require a lot of dependent serial

computation and high memory requirement. On the other hand, general-purpose architectures such as CPU or GPU cannot perform RNNs efficiently, because they limit the amount of parallelism that can be exploited due to the data dependencies. Therefore, accelerating LSTM networks through either customized architectures or FPGA for ASR systems is one the most challenging and interesting future paths.

As aforementioned, in order for the ASR system to provide high accuracy, it would require several integrated components regarding the acoustic-scoring, graph search and rescoring phases. Each of these stages are characterized with specific properties and unique computational behaviour, which require their own customized hardware architecture in order to achieve a high-performance, low-power and high memory efficiency. Thus, we believe a heterogeneous acceleration design targeted for small-form battery-operated devices such as smart phones, is required to obtain the best trade-off in the sense of real-time performance, energy and area cost.

Furthermore, there is not equal complexity for decoding the speech when processing different utterances. Therefore, it can be an interesting future line of research to design an adaptable hardware architecture for ASR systems, which can detect the difficulty of speech using either run-time feedback or from some properties of the input audio signal. In this thesis, we have explored some metrics like the frame-to-frame data locality and confidence of search to efficiently handle the Viterbi search space. However, some other techniques could potentially skip the DNN computations at some frames while using the previous scores with some small error-correcting mechanism, or checking whether the search result can predict any other feedback that could be useful for managing ASR workload.

Bibliography

[1] Apple's Network to detect "HeySiri". https://machinelearning.apple.com/2017/10/01/hey-siri.html.

[2] Google Now. https://en.wikipedia.org/wiki/Google_Now.

[3] Kaldi ASR results. https://github.com/kaldi-asr/kaldi/blob/master/egs/librispeech/s5/RESULTS.

[4] NVIDIA Tegra X1. http://international.download.nvidia.com/pdf/tegra/Tegra-X1-whitepaper-v1.0.pdf.

[5] Synopsys. https://www.synopsys.com/. Accessed: 2017-07-20.

[6] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Y. Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin. *CoRR*, abs/1512.02595, 2015.

[7] X. Anguera, S. Bozonnet, N. Evans, C. Fredouille, G. Friedland, and O. Vinyals. Speaker diarization: A review of recent research. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(2):356–370, Feb 2012.

[8] S. Anwar, K. Hwang, and W. Sung. Fixed point optimization of deep convolutional neural networks for object recognition. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1131–1135, April 2015.

[9] https://en.wikipedia.org/wiki/Siri.

[10] https://en.wikipedia.org/wiki/Softmax_function.

[11] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[12] L Bahl, Raimo Bakis, P Cohen, A Cole, Frederick Jelinek, B Lewis, and R Mercer. Further results on the recognition of a continuously read natural corpus. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'80.*, volume 5, pages 872–875. IEEE, 1980.

[13] Jon Barker, Shinji Watanabe, Emmanuel Vincent, and Jan Trmal. The fifth 'chime' speech separation and recognition challenge: Dataset, task and baselines. *CoRR*, abs/1803.10609, 2018.

[14] Eric Battenberg, Jitong Chen, Rewon Child, Adam Coates, Yashesh Gaur, Yi Li, Hairong Liu, Sanjeev Satheesh, David Seetapun, Anuroop Sriram, and Zhenyao Zhu. Exploring neural transducers for end-to-end speech recognition. *CoRR*, abs/1707.07413, 2017.

[15] Alain Biem, Erik McDermott, and Shigeru Katagiri. A discriminative filter bank model for speech recognition. In *EUROSPEECH*, 1995.

[16] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 111–118, USA, 2010. Omnipress.

[17] D. Caseiro and I. Trancoso. Transducer composition for "on-the-fly" lexicon and language model integration. In *Automatic Speech Recognition and Understanding, 2001. ASRU '01. IEEE Workshop on*, pages 393–396, 2001.

[18] Diamantino Caseiro and Isabel Trancoso. On integrating the lexicon with the language model, 2001.

[19] Andre Xian Ming Chang, Berin Martini, and Eugenio Culurciello. Recurrent neural networks hardware implementation on fpga. *CoRR*, abs/1511.05552, 2015.

[20] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam. Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622, Dec 2014.

[21] J. Choi, K. You, and W. Sung. An fpga implementation of speech recognition with weighted finite state transducers. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1602–1605, March 2010.

[22] Jike Chong, Ekaterina Gonina, and Kurt Keutzer. Efficient automatic speech recognition on the gpu. Chapter in GPU Computing Gems Emerald Edition, Morgan Kaufmann, 2011.

[23] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[24] Christopher Cieri, David Miller, and Kevin Walker. The fisher corpus: a resource for the next generations of speech-to-text. In *LREC*, 2004.

[25] Christopher Cieri, David Miller, and Kevin Walker. The fisher corpus: a resource for the next generations of speech-to-text. In *LREC*, volume 4, pages 69–71, 2004.

[26] Bernard Cole. Micron shows off memory muscle with new iot/m2m mcps. *Micron Technology*, November 2014.

[27] VoxForge Speech Corpus. http://www.voxforge.org. 2009.

[28] Alberto Delmas, Sayeh Sharify, Patrick Judd, and Andreas Moshovos. Tartan: Accelerating fully-connected and convolutional layers in deep learning networks by exploiting numerical precision variability. *CoRR*, abs/1707.09068, 2017.

[29] Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pages 2148–2156, 2013.

[30] K. Flautner, Nam Sung Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. In *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*, pages 148–157, 2002.

[31] M. Friesen. Linux power management optimization on the nvidia jetson platform. Technical report, 2016.

[32] John W. C. Fu, Janak H. Patel, and Bob L. Janssens. Stride directed prefetching in scalar processors. In *Proceedings of the 25th Annual International Symposium on Microarchitecture*, MICRO 25, pages 102–110, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[33] GeForce 900 series. https://en.wikipedia.org/wiki/GeForce_900_series.

[34] J. J. Godfrey, E. C. Holliman, and J. McDaniel. Switchboard: telephone speech corpus for research and development. In *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 517–520 vol.1, March 1992.

[35] David Graff. The 1996 broadcast news speech and language-model corpus. In *Proceedings of the 1997 DARPA Speech Recognition Workshop*, pages 11–14, 1996.

[36] A. Graves, A. r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013.

[37] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1764–1772, Bejing, China, 22–24 Jun 2014. PMLR.

[38] Reinhold Haeb-Umbach and Hermann Ney. Linear discriminant analysis for improved large vocabulary continuous speech recognition. In *Proceedings of Acoustics, Speech, and Signal Processing*, pages 13–16, 1992.

[39] 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 *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 243–254, June 2016.

[40] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, Huazhong Yang, and William J. Dally. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *FPGA*, 2017.

[41] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[42] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1135–1143. Curran Associates, Inc., 2015.

[43] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[44] Awni Y Hannun, Andrew L Maas, Daniel Jurafsky, and Andrew Y Ng. First-pass large vocabulary continuous speech recognition using bi-directional recurrent dnns. *arXiv preprint arXiv:1408.2873*, 2014.

[45] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[46] T. Hori, C. Hori, Y. Minami, and A. Nakamura. Efficient wfst-based one-pass decoding with on-the-fly hypothesis rescoring in extremely large vocabulary continuous speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(4):1352–1365, May 2007.

[47] R. Hsiao, M. Fuhs, Q. J. Y. Tam, I. Lane, and T. Schultz. Handbook of natural language processing and machine translation. page 496–504, 2011.

[48] David Huggins-Daines, Mohit Kumar, Arthur Chan, Alan W Black, Mosur Ravishankar, and Alexander I Rudnicky. Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 1, pages I–I. IEEE, 2006.

[49] K. Hwang and W. Sung. Fixed-point feedforward deep neural network design using weights +1, 0, and -1. In *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 1–6, Oct 2014.

[50] Homan Igehy, Matthew Eldridge, and Kekoa Proudfoot. Prefetching in a texture cache architecture. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, HWWS '98, pages 133–ff., New York, NY, USA, 1998. ACM.

[51] Adam Louis Janin. *Speech recognition on vector architectures.* PhD thesis, University of California, Berkeley, 2004.

[52] J. R. Johnston and R. A. Rutenbar. A high-rate, low-power, asic speech decoder using finite state transducers. In *2012 IEEE 23rd International Conference on Application-Specific Systems, Architectures and Processors*, pages 77–85, July 2012.

[53] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos. Stripes: Bit-serial deep neural network computing. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12, Oct 2016.

[54] Patrick Judd, Jorge Albericio, Tayler H. Hetherington, Tor M. Aamodt, Natalie D. Enright Jerger, Raquel Urtasun, and Andreas Moshovos. Reduced-precision strategies for bounded memory in deep neural nets. *CoRR*, abs/1511.05236, 2015.

[55] Stephan Kanthak, Hermann Ney, Michael Riley, and Mehryar Mohri. A comparison of two lvr search optimization techniques. In *IN PROC. INT. CONF. SPOKEN LANGUAGE PROCESSING*, pages 1309–1312, 2002.

[56] K.-F. Lee, H.-W. Hon, and R. Reddy. An overview of the sphinx speech recognition system. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 38(1):35–45, Jan 1990.

[57] M. Lee, K. Hwang, J. Park, S. Choi, S. Shin, and W. Sung. Fpga-based low-power speech recognition with recurrent neural networks. In *2016 IEEE International Workshop on Signal Processing Systems (SiPS)*, pages 230–235, Oct 2016.

[58] Edward C. Lin and Rob A. Rutenbar. A multi-fpga 10x-real-time high-speed search engine for a 5000-word vocabulary speech recognizer. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '09, pages 83–92, New York, NY, USA, 2009. ACM.

[59] Xie Lingyun and Du Limin. Efficient viterbi beam search algorithm using dynamic pruning. In *Signal Processing, 2004. Proceedings. ICSP '04. 2004 7th International Conference on*, volume 1, pages 699–702 vol.1, Aug 2004.

[60] Andrej Ljolje, Fernando Pereira, and Michael Riley. Efficient general lattice generation and rescoring. In *EUROSPEECH*, 1999.

[61] Andrew Lumsdaine, Douglas Gregor, Bruce Hendrickson, and Jonathan Berry. Challenges in parallel graph processing. *Parallel Processing Letters*, 17(01):5–20, 2007.

[62] Charith Mendis, Jasha Droppo, Saeed Maleki, Madanlal Musuvathi, Todd Mytkowicz, and Geoffrey Zweig. Parallelizing wfst speech decoders. In *Proceedings of Acoustics, Speech and Signal Processing*, pages 5325–5329, 2016.

[63] Yajie Miao, Mohammad Gowayyed, and Florian Metze. Eesen: End-to-end speech recognition using deep rnn models and wfst-based decoding. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 167–174. IEEE, 2015.

[64] Micron. Nor — nand flash guide. *Micron Technology*, 2017.

[65] https://en.wikipedia.org/wiki/Cortana_(software).

[66] Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, 16(1):69 – 88, 2002.

[67] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. Cacti 6.0: A tool to model large caches. *HP Laboratories*, pages 22–31, 2009.

[68] U. Nallasamy, I. Lane, M. Fuhs, M. Noamany, Y. Tam, Q. Jin, and T. Schultz. Handbook of natural language processing and machine translation. page 535–540, 2011.

[69] K. J. Nesbit and J. E. Smith. Data cache prefetching using a global history buffer. In *Software, IEE Proceedings-*, pages 96–96, Feb 2004.

[70] K. Nii, H. Makino, Y. Tujihashi, C. Morishima, Y. Hayakawa, H. Nunogami, T. Arakawa, and H. Hamano. A low power sram using auto-backgate-controlled mt-cmos. In *Low Power Electronics and Design, 1998. Proceedings. 1998 International Symposium on*, pages 293–298, Aug 1998.

[71] NVIDIA Visual Profiler. https://developer.nvidia.com/nvidia-visual-profiler.

[72] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. Librispeech: An asr corpus based on public domain audio books. In *ICASSP*, 2015.

[73] Douglas B. Paul and Janet M. Baker. The design for the wall street journal-based csr corpus. In *Proceedings of the Workshop on Speech and Natural Language*, HLT '91, pages 357–362, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[74] Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. A time delay neural network architecture for efficient modeling of long temporal contexts. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[75] Daniel Povey. Kaldi software. http://kaldi-asr.org/. Accessed: 2017-07-20.

[76] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*, number EPFL-CONF-192584. IEEE Signal Processing Society, 2011.

[77] Michael Powell, Se-Hyun Yang, Babak Falsafi, Kaushik Roy, and T. N. Vijaykumar. Gated-vdd: A circuit technique to reduce leakage in deep-submicron cache memories. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, ISLPED '00, pages 90–95, New York, NY, USA, 2000. ACM.

[78] 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*, 50(1):102–112, Jan 2015.

[79] M. Price, J. Glass, and A. P. Chandrakasan. A low-power speech recognizer and voice activity detector using deep neural networks. *IEEE Journal of Solid-State Circuits*, 53(1):66–75, Jan 2018.

[80] Michael Price. Energy-scalable speech recognition circuits (doctoral dissertation). In *Massachusetts Institute of Technology*, 2016.

[81] Michael Price, Anantha Chandrakasan, and James R. Glass. Memory-efficient modeling and search techniques for hardware asr decoders. In *INTERSPEECH*, pages 1893–1897, 2016.

[82] P. Price, W. M. Fisher, J. Bernstein, and D. S. Pallett. The darpa 1000-word resource management database for continuous speech recognition. In *ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing*, pages 651–654 vol.1, April 1988.

[83] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, Feb 1989.

[84] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[85] Mosur K Ravishankar. *Efficient Algorithms for Speech Recognition.* PhD thesis, 1996.

[86] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 267–278, Piscataway, NJ, USA, 2016. IEEE Press.

[87] M. Riera, J. Arnau, and A. Gonzalez. Computation reuse in dnns by exploiting input similarity. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 57–68, June 2018.

[88] Anthony Rousseau, Paul Deléglise, and Yannick Estève. Enhancing the ted-lium corpus with selected data for language modeling and more ted talks. In *In Proc. LREC*, pages 26–31, 2014.

[89] Albert Segura Salvador. Characterization of speech recognition systems on gpu architectures (master dissertation). In *Universitat Politecnica de Cataluya*, 2016.

[90] Johan Schalkwyk, Doug Beeferman, Françoise Beaufays, Bill Byrne, Ciprian Chelba, Mike Cohen, Maryam Kamvar, and Brian Strope. *"Your Word is my Command": Google Search by Voice: A Case Study*, pages 61–90. Springer US, Boston, MA, 2010.

[91] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.

[92] Franyell Silfa, Gem Dot, Jose-Maria Arnau, and Antonio Gonzàlez. E-pur: An energy-efficient processing unit for recurrent neural networks. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, PACT '18, pages 18:1–18:12, New York, NY, USA, 2018. ACM.

[93] https://www.skype.com/en/features/skype-translator/.

[94] Skype Translator. https://en.wikipedia.org/wiki/Skype_Translator.

[95] James E. Smith. Decoupled access/execute computer architectures. *ACM Trans. Comput. Syst.*, 2(4):289–308, November 1984.

[96] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[97] Hamid Tabani. *Low-power architectures for automatic speech recognition.* PhD thesis, Universitat Politecnica de Catalunya (UPC), 2018.

[98] Hamid Tabani, Jose-Maria Arnau, Jordi Tubella, and Antonio González. Performance analysis and optimization of automatic speech recognition. *IEEE Transactions on Multi-Scale Computing Systems*, 2017.

[99] Hamid Tabani, Jose-Maria Arnau, Jordi Tubella, and Antonio Gonzalez. An ultra low-power hardware accelerator for acoustic scoring in speech recognition. In *Parallel Architecture and Compilation Techniques (PACT), 26th International Conference on*. IEEE/ACM, 2017.

[100] TN-41-01. Calculating memory system power for ddr3, micron technology, tech. rep. Technical report, 2007.

[101] TN-53-01. Lpddr4 power calculator, micron technology, tech. rep. Technical report, 2016.

[102] K. Tokuda, T. Yoshimura, T. Masuko, T. Kobayashi, and T. Kitamura. Speech parameter generation algorithms for hmm-based speech synthesis. In *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.00CH37100)*, volume 3, pages 1315–1318 vol.3, June 2000.

[103] R. Vergin, D. O'Shaughnessy, and A. Farhat. Generalized mel frequency cepstral coefficients for large-vocabulary speaker-independent continuous-speech recognition. *Speech and Audio Processing, IEEE Transactions on*, 7(5):525–532, Sep 1999.

[104] V.M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore. Measuring energy and power with papi. In *ICPPW*, pages 262–268, 2012.

[105] W. Xiong, L. Wu, Fil Alleva, Jasha Droppo, Xuedong Huang, and Andreas Stolcke. The microsoft 2017 conversational speech recognition system. *CoRR*, abs/1708.06073, 2017.

[106] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. Achieving human parity in conversational speech recognition. *CoRR*, abs/1610.05256, 2016.

[107] R. Yazdani, M. Riera, J. Arnau, and A. González. The dark side of dnn pruning. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 790–801, June 2018.

[108] Reza Yazdani, Jose-Maria Arnau, and Antonio González. Unfold: a memory-efficient speech recognizer using on-the-fly wfst composition. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 69–81. ACM, 2017.

[109] Reza Yazdani, Albert Segura, Jose-Maria Arnau, and Antonio Gonzalez. An ultra low-power hardware accelerator for automatic speech recognition. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pages 1–12. IEEE, 2016.

[110] Reza Yazdani, Albert Segura, Jose-Maria Arnau, and Antonio Gonzalez. An ultra low-power hardware accelerator for automatic speech recognition. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12, Oct 2016.

[111] Reza Yazdani, Albert Segura, Jose-Maria Arnau, and Antonio Gonzalez. Low-power automatic speech recognition through a mobile gpu and a viterbi accelerator. *IEEE Micro*, 37(1):22–29, 2017.

[112] Kisun You, Jungwook Choi, and Wonyong Sung. Flexible and expandable speech recognition hardware with weighted finite state transducers. *Journal of Signal Processing Systems*, 66(3):235–244, Mar 2012.

[113] Kisun You, Jike Chong, Youngmin Yi, Ekaterina Gonina, Christopher J Hughes, Yen-Kuang Chen, Wonyong Sung, and Kurt Keutzer. Parallel scalability in speech recognition. *Signal Processing Magazine, IEEE*, 26(6):124–135, 2009.

[114] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. Scalpel: Customizing dnn pruning to the underlying hardware parallelism. In *Proceedings of ISCA*, pages 548–560, 2017.

[115] X. Zhang, J. Trmal, D. Povey, and S. Khudanpur. Improving deep neural network acoustic models using generalized maxout networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 215–219, May 2014.

[116] Xiaohui Zhang, Jan Trmal, Daniel Povey, and Sanjeev Khudanpur. Improving deep neural network acoustic models using generalized maxout networks. In *Proceedings of Acoustics, Speech and Signal Processing*, pages 215–219, 2014.

Glossary

Acoustic Model An acoustic model is used in Automatic Speech Recognition to represent the relationship between an audio signal and the phonemes or other linguistic units that make up speech. The model is learned from a set of audio recordings and their corresponding transcripts.

ASR. Automatic Speech Recognition, the conversion of a speech signal to a symbolic representation by computational means.

Beam. In a heuristic search algorithm, the beam is the range of scores outside of which current states or paths will not be extended. This is usually done by applying a factor to the best score of all paths reaching a given point and discarding all paths whose scores are less than the resulting score.

Continuous Acoustic Model. An HMM-based acoustic model, whose output density functions are Gaussian Mixture Models which each triphone has its own separate weighted Gaussian distributions.

Decoding. Translation of a message into codewords. In the case of speech, refers to the translation from a representation of a speech signal into a sequence of words or linguistic units. Usually, but not always, the input representation is the acoustic signal (for example, we can also speak of decoding a word lattice). Often used synonymously with search.

HMM. Hidden Markov Model. A statistical model used to exemplify a process which evolves over time, where the exact state of the process is unknown, or "hidden".

Hypothesis. A single sequence of words or linguistic units considered by a speech recognizer as the result of decoding an input utterance. May also refer to one particular component of such a sequence, as in a "word hypothesis".

Lattice. A directed acyclic graph representation of the set of hypotheses generated by a speech recognizer, where both word identities and timing information are represented.

MFCC. Mel-Frequency Cepstral Coefficients, the coefficients of the cepstrum of the short-term spectrum, downsampled and weighted according to the mel scale, a frequency scale thought to represent the sensitivity of the human ear.

Real-time factor. Often abbreviated as xRT, the measure typically used to report the performance of a speech recognizer. This is calculated as the ratio between the amount of time required to decode an utterance and the length of the utterance. For example, a real-time factor of 0.4 xRT means that each second of audio requires 0.4 seconds to decode (lower RTF means faster decoding).

Search. In automatic speech speech recognition, the process of searching the set of linguistic or symbolic representations of an utterance for one (or more) which are considered the most probable by the statistical models used by the recognizer. Often used synonymously with decoding.

Senone. The two most common forms of parameter tying are state tying and mixture tying. In the former, the state output distributions of all triphones are mapped to a set of acoustic clusters, usually known as senones. Each senone consists of a complete mixture distribution with its own set of Gaussian parameters.

Triphone. In large-vocabulary continuous speech recognition, it is common practice to use context-dependent sub-word units as the basic units of recognition. These are typically phoneme-like units, consisting of a single phoneme in a particular phonetic context. Most frequently, the context is defined by the identities of the phonemes immediately preceding and following, in order to model coarticulatory effects. This unit is known as a triphone.

Utterance. The longest segment of speech operated on at one time by an automatic speech recognizer. Typically corresponds to an uninterrupted phrase, sentence, or paragraph spoken by a single speaker.

Viterbi Algorithm. A dynamic programming algorithm to find the probability of the most likely state sequence.

Deep-Neural-Network (DNN). A machine-learning primitive which includes of one input and one output layer of neurons plus several hidden layers, that can be trained with large amount of data to do some classification based on a particular problem.

Recurrent-Neural-Network (RNN). A machine-learning primitive which contains of several layers each composed of one RNN cell, that recurrently processes an input sequence while storing the past states of the information to be used in the next or past evaluation of the network.

WER (Word Error Rate). Number of insertions plus deletions plus substitutions that are required to convert the recognized word sequence into the reference word sequence, divided by the total number of words of the reference utterance.