An Ultra Low-power Hardware Accelerator for Acoustic Scoring in Speech Recognition

Hamid Tabani, Jose-Maria Arnau, Jordi Tubella, Antonio González
Department of Computer Architecture, Universitat Politècnica de Catalunya
$\{htabani, jarnau, jordit, antonio\}$@$ac.upc.edu$

***Abstract*—Accurate, real-time Automatic Speech Recognition (ASR) comes at a high energy cost, so accuracy has often to be sacrificed in order to fit the strict power constraints of mobile systems. However, accuracy is extremely important for the end-user, and today's systems are still unsatisfactory for many applications. The most critical component of an ASR system is the acoustic scoring, as it has a large impact on the accuracy of the system and takes up the bulk of execution time. The vast majority of ASR systems implement the acoustic scoring by means of Gaussian Mixture Models (GMMs), where the acoustic scores are obtained by evaluating multidimensional Gaussian distributions.**

In this paper, we propose a hardware accelerator for GMM evaluation that reduces the energy required for acoustic scoring by three orders of magnitude compared to solutions based on CPUs and GPUs. Our accelerator implements a lazy evaluation scheme where Gaussians are computed on demand, avoiding 50% of the computations. Furthermore, it employs a novel clustering scheme to reduce the size of the acoustic model, which results in 8x memory bandwidth savings with a negligible impact on accuracy. Finally, it includes a novel memoization scheme that avoids 74.88% of floating-point operations. The end design provides a 164x speedup and 3532x energy reduction when compared with a highly-tuned implementation running on a modern mobile CPU. Compared to a state-of-the-art mobile GPU, the GMM accelerator achieves 5.89x speedup over a highly optimized CUDA implementation, while reducing energy by 241x.

***Keywords*-Automatic Speech Recognition; Gaussian Mixture Model; Acoustic Scoring; Hardware Accelerator;**

I. Introduction

Automatic Speech Recognition (ASR) is becoming a key technology for mobile devices [2], [7], [4], significantly changing the way people interact with computers. Voice-based user interfaces require accurate, large-vocabulary, speaker-independent, real-time ASR. Unfortunately, supporting all these capabilities comes at a high computational cost, which is very challenging to achieve in tightly energy constrained platforms such as smartphones, tablets or wearable devices.

Figure 1 shows decoding time per second of speech, memory footprint and accuracy, i. e. Word Error Rate (WER), for three popular open-source ASR systems when running on an ARM Cortex-A57. Pocketsphinx [19] uses Gaussian Mixture Models (GMMs) for acoustic scoring, whereas Kaldi [28] and Eesen [25] employ a Deep Neural Network (DNN) and a Recurrent Neural Network (RNN) respectively. We use the most recent pre-trained acoustic models provided in their respective websites [1]. As it can be seen in Figure 1, Pocketsphinx achieves much lower decoding time than the DNN-based systems, at the cost of a small increase in WER. Furthermore, it exhibits a small memory footprint of 85 MB, whereas Kaldi and Eesen require hundreds of MB. These numbers suggest that Pocketsphinx is the most suited approach for ultra low-power, low-cost devices, due to its lower computational and storage demands, so we chose it as the baseline for our design.

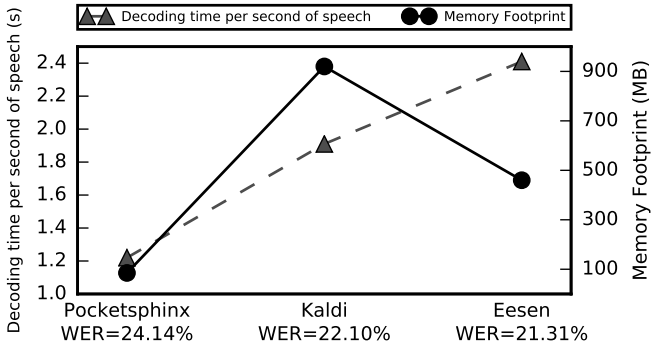


Figure 1: Decoding time, memory footprint and Word Error Rate (WER) for three popular ASR toolkits running on an ARM Cortex-A57. Results are collected using the most recent pre-trained models for each system.

The most compute intensive component of the majority of ASR systems, including Pocketsphinx, is the acoustic scoring. In an ASR system, the input audio signal is split in frames, typically of 10 ms duration. For each frame of speech, the acoustic scoring computes the likelihood, a.k.a. score, that the frame is part of a particular phoneme, for all potential phonemes in the language. Despite the increasing popularity of DNNs, GMMs are used by the majority of ASR systems including Pocketsphinx, Sphinx4, HTK [34], Julius [21] and some decoders of Kaldi by the reasons given above. GMM evaluation is typically the main bottleneck in these systems. Our measurements on an NVIDIA Tegra

[1]Higher accuracy can be achieved with even larger acoustic models and multiple rescoring passes [5], but they result in much larger memory footprint, energy consumption and decoding time.

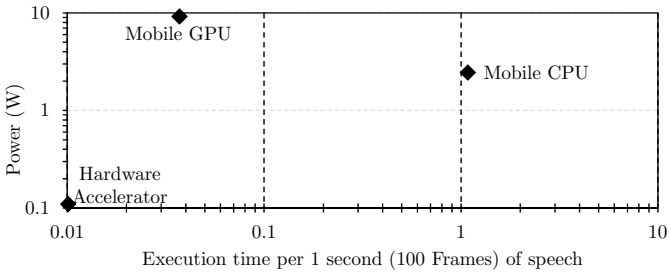


Figure 2: Power dissipation vs execution time for a mobile CPU, ARM Cortex-A57, a mobile GPU, NVIDIA Tegra X1, and a hardware accelerator for acoustic scoring.

X1 show that acoustic scoring takes more than 80% of the decoding time in Pocketsphinx.

Figure 2 shows the power dissipation and execution time for the GMM evaluation stage, i. e. acoustic scoring, on a mobile CPU and GPU. These numbers were collected using the standard acoustic model of Pocketsphinx, that requires the evaluation of 160k 36-dimensional Gaussian distributions for every frame of speech (10 ms). The mobile CPU dissipates 2.4 W, and it barely reaches real-time performance as it takes 1.08 s to process one second of speech. The mobile GPU achieves real-time performance, but at the cost of increasing power dissipation significantly (9.2 W), which in turn results in shorter operating times per battery charge. To illustrate this issue, for a typical smartphone battery of 11.55 WHr (41580 J) [8] the operating time when running ASR software on the CPU would be less than 4.8 hours, whereas it would be reduced to less than 1.25 hours when using the GPU. Note that CPU and GPU are not the only battery consumers in a smartphone, so these numbers are upper bounds of the operating time.

Hardware acceleration is an effective approach to achieve high-performance and low-power ASR in mobile devices. In this paper, we present an accelerator for GMM computation that includes innovative techniques to improve the energy-efficiency of ASR systems. Our accelerator implements a lazy evaluation scheme that computes Gaussian distributions on demand instead of evaluating all the Gaussians on every frame, reducing the amount of computation. Our lazy evaluation scheme introduces a novel technique to predict the active Gaussians in the next frames of speech, in order to apply lazy evaluation for multiple frames (batching).

On the other hand, our accelerator includes a novel clustering technique that provides 8x reduction in the size of the acoustic model. Our clustering works independently for each dimension of the Gaussian distributions, and we show that it provides better accuracy than the straightforward approach of clustering the entire GMM together.

Finally, we show that as a side effect, the proposed clustering results in a huge amount of redundant computations, and we introduce a novel memoization scheme that drastically reduces the number of floating point operations. Our scheme pre-computes and stores in hardware all possible results at the beginning of each frame, instead of probing/updating a memoization table for every individual floating point operation. Figure 2 shows the power and performance of our GMM accelerator. As it can be seen, the accelerator achieves higher performance than the mobile GPU at ultra low power, as low as 110 mW.

Our accelerator supports any acoustic model based on Gaussian Mixture Models, so it works for speech in any language and it supports the acoustic models available in Pocketsphinx, Kaldi, Julius or HTK. Moreover, we design our accelerator with enough throughput and storage to support larger and more accurate acoustic models that are likely to appear in the next years. We believe speech recognition will be a feature supported by the majority of computing devices in the near future, and acoustic models will evolve towards more complex ones for the sake of better accuracy, which will put more pressure on energy-efficient solutions.

Note that GMM is a popular machine learning algorithm also used in different areas. The proposed GMM accelerator can be used for other applications that are relevant for mobile devices, especially in the area of computer vision. GMMs are employed for image segmentation [16], [17], image retrieval [31], tracking people in images or detecting and tracking moving objects in video sequences [12].

This paper focuses on ultra low-power, high-performance ASR. Its main contributions are the following:

- We analyze the performance and energy consumption of ASR on a state-of-the-art mobile CPU and GPU.
- We propose a hardware accelerator for GMM evaluation, which is the main bottleneck of ASR systems.
- We develop a lazy computation scheme that evaluates Gaussian distributions on demand, reducing the amount of computation by more than 50%. We show how lazy evaluation can be combined with batch processing of multiple frames by predicting active Gaussians.
- We introduce a novel clustering scheme to reduce the size of GMM parameters. A thorough analysis of the impact of clustering in accuracy, power and performance is presented, in order to select the most efficient configuration. The proposed scheme provides 8x memory bandwidth savings with a negligible impact on accuracy.
- We show that the use of clustering increases the degree of redundancy, and propose a novel memoization scheme that removes 74.88% of the floating-point operations.
- Overall, our GMM accelerator provides 164x speedup and 3532x energy reduction with respect to the mobile CPU of a Tegra X1. Compared to the mobile GPU of a Tegra X1, the GMM accelerator achieves 5.89x speedup and 241x energy reduction.

The remainder of the paper is organized as follows. Section II provides background on ASR systems and acoustic

scoring. Section III presents the basic accelerator design and the lazy evaluation scheme. Section IV introduces the clustering techniques and the memoization scheme. Section V describes the evaluation methodology, whereas the experimental results are provided in Section VI. Section VII reviews related work and, finally, Section VIII sums up our conclusions.

II. Background

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 stages: *Feature Extraction*, *Acoustic Scoring* and *Search Engine*. This pipeline works as follows. 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* component transforms the audio samples within a frame into a vector of features. These features are then converted into a sequence of phonemes by the *Acoustic Scoring*. Finally, the *Search Engine* transforms the sequence of phonemes into a sequence of words by performing a Viterbi beam search [29], [33], [32].

The *Acoustic Scoring* is known to be the most expensive part of an ASR system [22], [18]. Our power/performance analysis of Pocketsphinx on an NVIDIA Tegra X1 also supports this claim, as the *Acoustic Scoring* takes more than 80% of the execution time.

A. Acoustic Scoring

ASR systems create acoustic models for sub-word units or phonemes. Due to co-articulation effects, the production of sound corresponding to a phoneme is influenced by neighboring phonemes. Context-dependent phones or triphones [9] are typically employed by ASR systems, as they can model such variations. There are around 50 phonemes in spoken English, so there are around 50^3 triphones, but only a fraction of them are observed in practice.

Triphones are modeled in most of the ASR systems by using Gaussian Mixture Models (GMM). The *g*-th component of the *m*-th GMM is a normal distribution with mean vector $\mu_{m,g}$ and standard deviation vector $\sigma_{m,g}$. Each mixture component also has a scalar mixture coefficient or weight $w_{m,g}$. Hence, the probability of observing a given frame of speech with feature vector x in the GMM m is given by Equation 1, where g ranges over the number of Gaussians in the mixture. The expression $\mathcal{N}(.)$ is the value of the Gaussian density function at x. For numerical stability, the multivariate Gaussian distribution $\mathcal{N}(.)$ is computed in log-space by using Equation 2. The dimensionality of the Gaussians is equal to the dimensionality of the feature vector x. During the recognition process Equation 1 has to be evaluated for every GMM on a frame basis.

$$GMM_m(x) = \sum_{g=0}^{N-1} w_{m,g}\mathcal{N}(x, \mu_{m,g}, \sigma_{m,g}) \qquad (1)$$

$$\mathcal{N}(x, \mu_{m,g}, \sigma_{m,g}) = D_{m,g} - \sum_{c=0}^{M-1} \frac{(x_c - \mu_{m,g,c})^2}{2\sigma^2_{m,g,c}} \qquad (2)$$

In a fully continuous acoustic model each triphone has its own separate weighted GMM. However, computing such a big number of Gaussian functions is completely unfeasible for real-time speech recognition. In practice, multiple triphones share the same GMM to reduce the computational cost of the *Acoustic Scoring*. Triphones are grouped into clusters called senones. All the triphones that belong to the same senone share the same GMM. Continuous acoustic models usually have a few thousand senones. For instance, the latest continuous acoustic model for English language provided by CMU Sphinx has 5000 senones.

Not all the senones are active for a given frame, as some senones may be pruned away during the search process. Some ASR systems, such as Pocketsphinx, only compute the GMM for active senones as the acoustic likelihood for the other, inactive, senones is not required for the search. By doing so, the workload for the continuous acoustic model is reduced by approximately one half. In order to implement this optimization the *Search Engine* generates a list of active senones based on the result of the pruning. The *Acoustic Scoring* uses this feedback to avoid GMM computation for senones that are inactive. This optimization not only reduces the complexity of computations, but it also reduces the accesses to main memory since the acoustic parameters for inactive senones do not need to be fetched.

III. Hardware Accelerated Acoustic Scoring

In this section, we present a high-performance and low-power accelerator for GMM evaluation, with the aim of improving the energy-efficiency of acoustic scoring in mobile ASR systems. The accelerator focuses on the GMM evaluation since it is the main performance and energy bottleneck. Our measurements on a NVIDIA Tegra X1 show that GMM evaluation takes more than 80% of the execution time of the whole ASR computation (in Pocketsphinx).

We first introduce a base design of the accelerator, that consists on a straightforward hardware implementation of the GMM evaluation method described in Section II. Next we present a lazy Gaussian evaluation scheme implemented on top of the base design, in order to avoid GMM computation for inactive senones.

A. GMM Accelerator

We first review the data and the computations required for GMM evaluation before presenting the architecture of the accelerator. GMM evaluation consists of computing Equation 2 for each multidimensional Gaussian distribution in the acoustic model. Regarding the data, each Gaussian requires multiple memory accesses to fetch the determinant ($D_{m,g}$), the input features (x_c), the means ($\mu_{m,g,c}$) and the variances ($\sigma^2_{m,g,c}$). To speed-up GMM computation the value

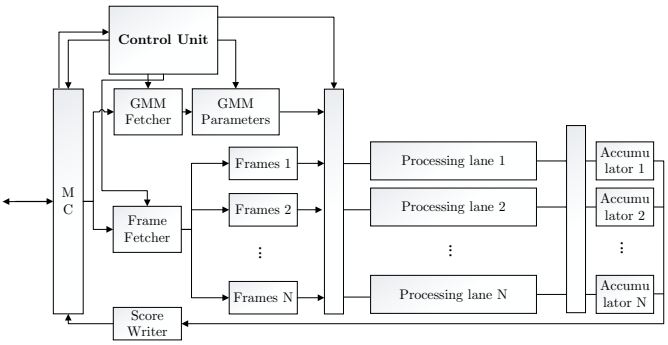


Figure 3: Architecture of the GMM accelerator.

$1/2\sigma^2_{m,g,c}$ is precomputed offline, as it does not depend on the input x_c, and fetched from memory instead of the variance. Therefore, processing one component of a Gaussian distribution requires one subtraction, two multiplications to compute the square and multiply the result by $1/2\sigma^2_{m,g,c}$, and one subtraction to perform the accumulation.

ASR systems typically compute GMM for batches of multiple frames in order to save memory bandwidth. Performing GMM evaluation for multiple frames in parallel improves temporal locality, as the means and variances can be fetched once from the off-chip system memory and reused for multiple frames, instead of fetching GMM parameters on a frame basis. Note that most of the memory bandwidth is employed for fetching Gaussians. For example, the acoustic model of Pocketsphinx for English language consists of 5000 mixtures or senones, where each mixture has 32 36-dimensional Gaussians. Therefore, the memory footprint is 44.55 MB: 21.9 MB for means, 21.9 MB for variances and 0.61 MB for the determinants, since parameters are stored as 32-bit floating point numbers.

Figure 3 shows the initial architecture of the GMM accelerator. Its pipeline works as follows. First, the *Frame Fetcher* unit is triggered to read the input features for all the frames in the batch from main memory. The features are stored in an internal SRAM memory with multiple banks named *Frames 1*, ..., *Frames N* in Figure 3. Typical ASR systems employ between 30 and 40 features per frame. We design our accelerator to support up to 64 features. Therefore, the size of each bank depends on the number of frames and the numbers of banks:

$$Bank\ Size = \frac{F\ frames \times 256\ bytes/frame}{B\ banks} \quad (3)$$

For example, for a batch size of 128 frames and 8 banks, the size of each bank is 4 KB. Once all the features have been fetched from main memory and stored in the on-chip SRAM, the *GMM Fetcher* unit is triggered to read the parameters for the first Gaussian. These parameters, i. e. means and variances, are stored in the *GMM parameters* SRAM memory. The accelerator is designed to support Gaussian distributions with up to 64 dimensions and, hence, one Gaussian requires up to 512 bytes of storage. The *GMM parameters* SRAM memory includes two banks of 512 bytes to store the current and the next Gaussian distribution. By using the two banks the accelerator can overlap the latency for fetching the next Gaussian with the computations for the current Gaussian.

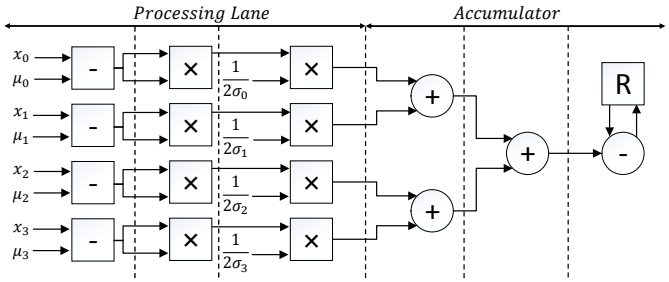


Figure 4: Pipeline of the *Processing lane* and *Accumulator* in the GMM accelerator.

Once the means and variances are available in the on-chip SRAM, the Gaussian is evaluated for all the frames in the batch by using the *Processing lanes* and the *Accumulators*. Figure 4 shows the pipeline for performing GMM computations. Each *Processing lane* includes SIMD FPUs to process four components of a Gaussian in parallel for one particular frame. The computation of the Gaussian works as follows. First, 4 means and 4 variances are fetched from the *GMM Parameters*, by performing one access of 256 bits, and broadcast them to the different *Processing lanes*. In parallel, a 128-bit fetch is performed in each of the *Frames 1 ... Frames N* SRAM memories to read 4 features for each frame. The *Processing lane* receives its 4 corresponding features and the 4 means and performs the first subtraction, $x_c - \mu_{m,g,c}$. Next, a SIMD FP multiplier is used to compute $(x_c - \mu_{m,g,c})^2$. In the next stage, another SIMD FP multiplier is used to multiply the previous result by the variance.

The *Accumulator* employs a reduction tree to compute the summation of the 4 components, as shown in Figure 4. Furthermore, it performs the final subtraction to update the score of the Gaussian. This process is repeated until the current Gaussian is evaluated for all the frames in the batch. Finally, the *Score Writer* unit stores the scores for the different frames in main memory.

The accelerator is able to overlap memory accesses with computations to a large extent. To this end, the processing of a Gaussian is split in three stages: fetching means and variances from memory, evaluating the Gaussian for all the frames in the batch and writing the scores in main memory. The three stages are pipelined, so the accelerator fetches the next Gaussian from memory while it evaluates the current Gaussian and writes the scores for the previous Gaussian.

The capacity of the accelerator to hide memory latency depends on the batch size as illustrated in Figure 5. This graph shows the time required for memory transfers and computations for 128 frames of speech in a version of the accelerator with 8 processing lanes, using the acoustic model of Pocketsphinx that contains 160k 36-dimensional

Figure 5: Execution time for memory transfers and computations for 128 frames of speech, using different batch sizes from 1 to 128.

Gaussians. As it can be seen, the memory transfer time is significantly reduced when increasing batch size, as the Gaussians are fetched once from main memory and reused for multiple frames. For a batch size bigger than 16 frames the memory transfers can be completely hidden with useful computations. Bigger batch sizes improve temporal locality and increase the FP to MEM ratio, improving the capacity of the accelerator to tolerate memory latency. However, it also increases the latency of the ASR system, as more frames of audio have to be buffered before the recognition process starts. For example, a batch size of 128 frames introduces a delay of 1.28 seconds (10 ms per frame). Therefore, the batch size cannot be arbitrarily increased for online speech recognition systems.

B. Lazy GMM Evaluation

The result generated by the GMM accelerator is consumed by the *Search Engine*, the next stage in the pipeline of an ASR system. The *Search Engine* employs the acoustic scores to perform a search on a graph-based recognition network, in order to find the sequence of words with maximum likelihood. For large vocabulary ASR it is unfeasible to explore the entire search space due to its huge size. ASR systems prune away paths that are very unlikely to match the input stream. Due to the pruning, the scores of some mixtures or senones are not used by the *Search Engine* and, hence, GMM computation for these inactive senones can be skipped. Figure 6 shows the number of active senones in several frames of speech.

Many ASR systems, such as Pocketsphinx, generate a list of active senones, i. e. senones whose acoustic scores are required to perform the search, for every frame of speech. We extend our GMM accelerator to access this list of active senones in order to reduce the amount of computation and the memory bandwidth usage. Instead of blindly computing all the senones, the GMM accelerator computes Gaussians on demand as required by the *Search Engine*. This version of the accelerator includes a SRAM memory to store the list of active senones. Assuming a batch size of F frames, this memory stores F bits for every senone. This bitmask indicates whether the senone is active on each one of the frames in the batch.

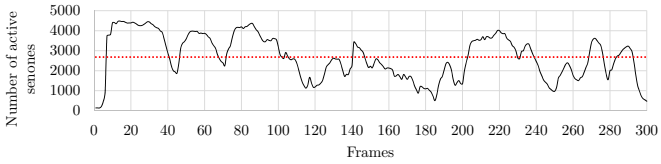


Figure 6: Number of active senones vs frames of speech. Red dotted line shows average number of active senones per frame.

A senone can only be completely skipped if the bitmask indicates that it is inactive in all the frames of a batch. In that case, means and variances for its N Gaussian distributions (e.g. 32 in Pocketsphinx) are not fetched from main memory. Moreover, all the associated GMM computations are avoided. However, if the senone is active in at least one frame then the GMM parameters have to be fetched from main memory. In that case, the accelerator can still avoid some computations by disabling the *Processing lanes* for frames where the senone is not active.

Figure 7 shows the percentage of active senones in Pocketsphinx when processing the test set of LibriSpeech corpus [27] (5.4 hours of speech). The bigger the batch size, the higher the likelihood that a senone is active in at least one frame. For a batch size of 128 frames more than 90% of the Gaussians have to be processed, but for modest batch sizes of 4-8 frames there is a large percentage of senones that are inactive in all the frames of the batch. Note that online ASR systems prefer small batch sizes to reduce the response time.

The memory bandwidth required for fetching the list of active senones is significantly smaller than the bandwidth used for fetching Gaussians. For example, for Pocketsphinx acoustic model (5000 senones) and a batch size of 8 the list of active senones contains 40k bits (4.8 KB). By fetching this information the accelerator is able to completely skip more than 40% of the senones (see Figure 7), so the lazy GMM evaluation scheme avoids fetching 17 MB from main memory per batch. Moreover, the amount of floating point computations is reduced by 57%.

Note that the list of active senones produced by the *Search Engine* is only available for the current frame and, hence, it is not clear how to apply lazy evaluation when processing frames in batches. For this reason, software solutions that implement lazy evaluation, such as Pocketsphinx, process frames sequentially, whereas systems that exploit batching have to compute all the Gaussians for every frame. In this paper, we propose a novel approach to combine the benefits of both lazy evaluation and batching. We observe that the speech signal is quasi-stationary when considering a short interval of time, and exploit this behavior to implement a simple prediction scheme: the accelerator predicts that the active senones in the next N - 1 frames will be the same as in the current frame, being N the batch size. This simple scheme achieves 94% and 83% of accuracy for batch sizes

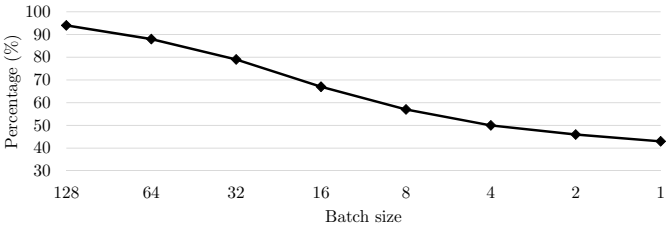


Figure 7: Percentage of active senones for different batch sizes. A senone is considered active if it is active in at least one of the frames of a batch.

of 3 and 8 frames respectively. We check the prediction with the list of active senones when it is available for each frame, and we only compute sequentially the senones that are mispredicted. For example, for a batch size of 8 only 17% of the senones are computed sequentially, whereas 83% employ batching. Therefore, our scheme avoids computing all the senones by using lazy evaluation, but it is still able to apply batching to a large extent, achieving substantial improvements in performance and energy consumption.

IV. Clustering and Memoization

Online ASR systems require small batch sizes of just a few frames of speech in order to achieve high responsiveness. Moreover, our lazy GMM evaluation scheme presented in Section III-B also requires small batches to skip a substantial percentage of the Gaussian distributions. The main disadvantage of small batch sizes is that the memory transfer time cannot be completely hidden by computations, as shown in Figure 5.

Most of the memory bandwidth in GMM evaluation is employed for fetching means and variances. For example, for Pocketsphinx acoustic model and a batch size of 8 frames, 21.9 MB of means and 21.9 MB of variances have to be fetched from memory for every batch. In comparison, only 0.61 MB, 1.12 KB and 4.8 KB have to be fetched for determinants, input features and list of active senones respectively. Therefore, 98.61% of the memory bandwidth is used for reading means and variances.

In this section, we first explore different alternatives for clustering the GMM parameters, with the aim of reducing the memory bandwidth usage for GMM evaluation. We show that the best clustering scheme provides 8x bandwidth reduction, with a negligible loss in accuracy. We next extend our GMM accelerator introduced in Section III to support clustered GMMs, to reduce its memory requirements and increase the overlap of memory accesses and computations. Finally, we show that the clustering scheme increases the amount of redundant computations to a large extent, and propose a memoization technique that avoids 74.88% of the floating point operations, which translates into important savings in energy consumption.

A. Clustering GMM Parameters

Reducing the size of the datasets for ASR is key for both performance and energy consumption. For small batch sizes used in online ASR systems, the performance of acoustic scoring is mainly constrained by memory transfers as illustrated in Figure 5. Furthermore, off-chip memory accesses are known to be one of the main sources of energy consumption in mobile devices and, hence, memory bandwidth reductions translate into important energy savings.

Compression algorithms can be used to achieve larger compression ratios, as long as they do not cause a significant decrease in accuracy. One of the most successful techniques for lossy GMM compression is clustering [30], [14], [10], K-means being the most popular algorithm. With clustering, the floating point values are replaced by significantly smaller integer indices into a codebook of centroids.

On the other hand, instead of clustering individual scalar values, vector clustering for GMM can be used as described in [30] to further increase compression ratio. With vector clustering multiple FP values are represented with just one index. Pocketsphinx acoustic model consists of 160k 36-dimensional Gaussians. Clustering entire Gaussians only requires one index for every 36 FP values. However, the bigger the length of the vector the higher the accuracy loss and, hence, larger codebooks are required to achieve accuracy levels close to the uncompressed dataset. Note that the codebook of centroids is also part of the compressed dataset and it must be considered for computing compression ratio. In order to achieve a better trade-off between compression ratio and accuracy, vector clustering can be implemented at the sub-Gaussian level, where each Gaussian is split in 2 18-dimensional vectors, or 3 12-dimensional vectors, etc.

We have implemented both vector and scalar clustering for GMM parameters and we have analyzed their impact on compression ratio and accuracy. The results are shown in Figure 8. Regarding vector clustering, we consider vector sizes of 4, 6, 9, 12 and 18. In addition, we change the number of clusters, i. e. the codebook size, between 1K and 12K. As it can be seen, for the same number of clusters the bigger the vector size the bigger the compression ratio, but the bigger the accuracy loss. Vector clustering achieves compression ratios close to 16x, but at the cost of a significant decrease in accuracy (more than 5% in absolute WER).

On the other hand, scalar clustering achieves better accuracy than vector quantization methods. Figure 8 depicts configurations using scalar clustering with K-means algorithm, for three different codebook sizes: 16 (“K-means_16”), 32 (“K-means_32”) and 256 centroids (“K-means_256”). The bigger the codebook size the smaller the accuracy loss, but the smaller the compression ratio as indices require more bits. The configuration with 256 clusters provides a reduction of the dataset close to 4x with a negligible impact on accuracy, 0.02% increase in WER. For a codebook of

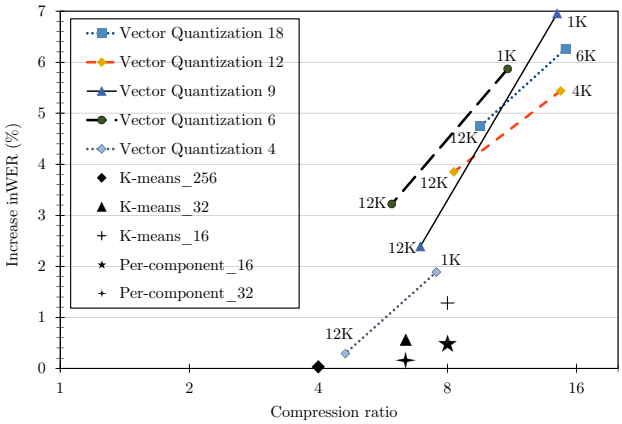


Figure 8: Compression ratio vs increase in Word Error Rate (WER) for different clustering algorithms.

16 centroids each index only requires 4 bits and, hence, compression ratio is close to 8x. However, WER is increased by more than 1.3%.

Increasing the number of clusters results in better accuracy, as the distances from the original values to the centroids of the clusters tend to be smaller. However, it also results in smaller compression ratio, since more bits are required to represent the indices. In this paper, we propose a novel strategy to increase the number of clusters without increasing the size of the indices. For an acoustic model with N Gaussians and C components per Gaussian, the dataset consists of an $N \times C$ matrix. In our scheme we apply clustering for each Gaussian component separately generating C different codebooks, one for each column. By doing this, the number of centroids is increased by a factor of C, but the index size remains the same as the size of each individual codebook is not increased. For example, assuming an acoustic model with 36-dimensional Gaussians and 16 clusters, we apply scalar K-means for each column of the matrix and generate 36 codebooks with 16 centroids, which means a total of 576 centroids for the entire dataset. Nevertheless, the index size is still 4 bits since each dimension of the Gaussian is restricted to its corresponding codebook. Note that the target codebook can be obtained from the column of the matrix, so indices in column c use codebook c and, hence, no additional information has to be stored to locate the codebook.

Figure 8 shows the results for our per-component clustering scheme, using 16 centroids per codebook ("Per-compon-ent_16") and 32 centroids ("Per-component_32"). Our scheme provides a better trade-off between compression ratio and accuracy than the schemes that apply K-means for the entire dataset with just one codebook. Compared to "K-means_256", "Per-component_16" doubles compression ratio, from 4x to 8x, despite increasing the number of clusters from 256 to 576, since each component is clustered separately and only 4 bits are required per index. Compared to "K-means_16", which also uses 4-bit indices, "Per-component_16" improves the increase in WER from 1.3% to 0.4% as it employs 576 centroids instead of 16 for the entire dataset. Therefore, per-component clustering achieves the same compression ratio as the schemes with a small number of clusters, while providing a level of accuracy close to the schemes with large codebooks.

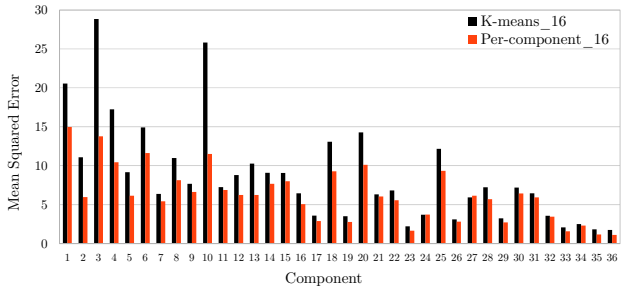


Figure 9: Mean Squared Error introduced by the clustering techniques for each component, or dimension, of the Gaussian distributions in Pocketsphinx English language acoustic model. Our per-component clustering provides smaller errors for most of the dimensions.

The per-component clustering generates highly tuned codebooks for each dimension. Figure 9 shows the Mean Squared Error (MSE) for each of the 36 dimensions in the acoustic model of Pocketsphinx, including our per-component clustering and the regular K-means with 16 clusters for the entire dataset. Per-component clustering introduces smaller errors for most of the dimensions, achieving higher accuracy with the same compression ratio.

We have implemented in our GMM accelerator the per-component clustering with codebooks of 16 centroids. This configuration provides close to 8x compression ratio: 32-bit FP values are replaced by 4-bit indices, whereas up to 4KB are used for codebooks (up to 64 dimensions multiplied by 16 centroids). For LibriSpeech test set (5.4 hours of audio) this configuration introduces a negligible error with respect to the uncompressed dataset, increasing WER by only 0.4%.

We extended our GMM accelerator to support clustering as illustrated in Figure 10. This version includes an on-chip SRAM memory, named *Centroids*, to store the codebooks. The accelerator supports Gaussians with up to 64 dimensions and codebooks with up to 16 centroids. The total size of the *Centroids* SRAM memory is 4 KB: 64 codebooks $\times$ 16 centroids/codebook $\times$ 4 bytes/centroid. The *Processing lanes* of the GMM accelerator require 4 means and 4 variances per cycle to achieve peak throughput. The *Centroids* memory is split in 4 banks of 1 KB with 2 read ports per bank, in order to fetch 4 centroids for means and 4 centroids for variances in one cycle. To avoid bank conflicts, codebooks for different components are interleaved with a factor of 4, as 4 consecutive components are processed per cycle.

When using clustering, the *GMM Fetcher* reads indices from main memory and stores them in the *GMM Parameters* SRAM memory. The pipeline of the accelerator includes an

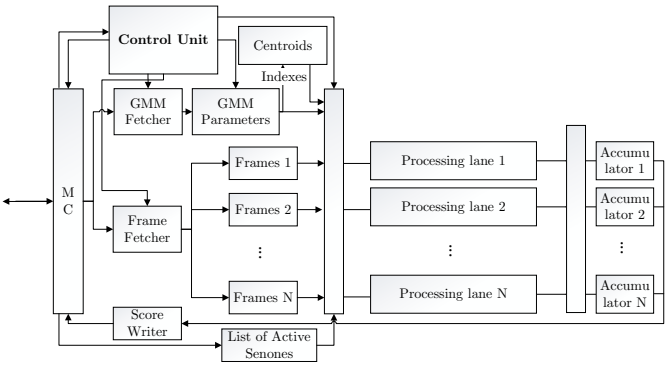


Figure 10: Architecture of the GMM accelerator including support for clustering and lazy evaluation.

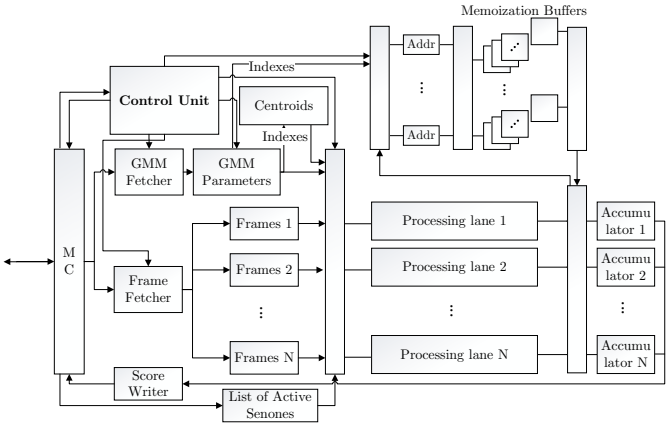


Figure 11: Architecture of the GMM accelerator including support for lazy evaluation, clustering and memoization.

additional stage to fetch the centroids. In this new stage, the indices for means and variances read from the *GMM Parameters* memory are used to obtain the corresponding values from the *Centroids* memory. These centroids are then dispatched to the *Processing lanes*, the rest of the pipeline remains unmodified and works as described in Section III.

The use of clustering provides a large reduction in memory bandwidth usage. For Pocketsphinx acoustic model, 43.8 MB are fetched from memory every batch to read means and variances when using the uncompressed dataset. Our clustering scheme reduces the size of the dataset to 5.5 MB, achieving a reduction in memory bandwidth close to 8x.

B. Memoizing GMM Computations

GMM evaluation for one frame of speech requires computing the expression $(x_c - \mu_{m,g,c})^2 \times (1/2\sigma^2_{m,g,c})$ for every component of every Gaussian (see Equation 2). The number of times the aforementioned expression is computed depends on the number of Gaussians in the acoustic model and the dimensionality of the Gaussians. For Pocketsphinx acoustic model, which contains 160K 36-dimensional Gaussians, this expression is evaluated 5.76 million times per frame. However, when using our clustering scheme described in Section IV-A, each one of the 36 input features (x_c) can only be combined with 16 different means and 16 variances. The total number of unique evaluations is $36 \times 16 \times 16 = 9216$. Therefore, 99.84% of the evaluations of this expression are redundant, i. e. they use the same value of x_c, $\mu_{m,g,c}$ and $\sigma^2_{m,g,c}$ as a previously computed component.

In this paper, we propose the use of memoization to avoid all these redundant computations. Memoization is an optimization technique that avoids repeating the execution of redundant computations by reusing the results of previous executions with the same input values. The first time a computation is executed, its result is dynamically cached in a Look Up Table (LUT). Subsequent executions of the same computation will obtain the result from the LUT rather than recalculating it.

We extend our GMM accelerator to memoize the result of the expression $(x_c - \mu_{m,g,c})^2 \times (1/2\sigma^2_{m,g,c})$, which corresponds to the first subtraction and the two multiplications performed in the *Processing lanes* as illustrated in Figure 4. In this version of the accelerator, we decouple the *Processing lanes* from the *Accumulators*. The *Processing lanes* are first triggered to precompute all the unique combinations of the aforementioned expression. The output results of the *Processing lanes* are stored in a new on-chip SRAM memory, named *Memoization Buffers*, as illustrated in Figure 11. This SRAM memory stores the unique results in a 4D matrix with dimensions: $Num\_frames\_in\_batch \times Num\_features\_per\_frame \times Num\_means \times Num\_variances$. Our accelerator supports up to 8 frames per batch, 64 features per frame and 16 centroids for means and variances, so the total size for the *Memoization Buffers* is 512 KB.

Once all the unique computations are executed, the scores of the different Gaussians are computed for the given batch of frames. Evaluating a Gaussian consists on performing a summation of the corresponding results stored in the *Memoization Buffers*, using the *Accumulators*. The indices for mean and variance, together with the frame index and feature index are used to access the 4D matrix of precomputed results. Two new pipeline stages are included to perform this access as illustrated in Figure 11. In the first stage, the address of the precomputed result is obtained from the different indices. In the next stage, this address is employed for accessing the *Memoization Buffers* in order to get the precomputed result for the expression $(x_c - \mu_{m,g,c})^2 \times (1/2\sigma^2_{m,g,c})$. The values obtained from the *Memoization Buffers* are forwarded to the *Accumulators*, where they are employed to update the score of the Gaussian.

In summary, GMM evaluation requires four FP operations for each component of every Gaussian. The first three FP operations exhibit a high degree of redundancy, since only 0.16% of the computations are unique whereas 99.84% are redundant. We extend our accelerator to precompute these 0.16% unique FP values and store them in a relatively small table, so they can be later used for computing Gaussians.

Technology / Frequency	28 nm / 1200 MHz
Number of lanes / Lane width	8 / 4
Parameter buffer	3 buffers, 1 KB each
Frame buffer	1 KB per each lane
Active senone / Centroids buffer	6 KB / 1 KB
Write score / Memoization buffer	1 KB / 512 KB
Functional units per lane	8 fp multipliers, 4 fp adders

Table I: Hardware parameters for the proposed accelerator.

In the base design of the accelerator, evaluating Pocketsphinx acoustic model for a batch size of 8 frames requires about 184M FP operations: $8\ frames \times 160K\ Gaussians \times 36\ components \times 4\ FP\ ops$. When using memoization, we first precompute the unique values, this requires about 221K FP operations: $8\ frames \times 36\ components \times 16\ means \times 16\ variances \times 3\ FP\ ops$. In addition, to get the scores we have to perform the summation of 36 of these precomputed values, which requires 46.08M FP ops: $8\ frames \times 160K\ Gaussians \times 36\ components \times 1\ FP\ op$. The total number of operations with the memoization scheme is 46.30M, i.e. 74.88% of the FP operations are avoided.

Computation re-use is lucrative only when the cost of accessing the structures used for memoization is smaller than the benefit of skipping the actual computation. According to our results obtained with CACTI and Synopsys Design Compiler at 28 nm, the energy for accessing one bank of the *Memoization Buffers* is much smaller than the cost of performing one FP subtraction and two FP multiplications.

V. Evaluation Methodology

We have developed a simulator that models the GMM accelerator described in Secion III. Furthermore, the simulator implements the lazy evaluation scheme presented in Section III-B, our clustering technique (see Section IV-A) and the memoization scheme (see Section IV-B). The parameters employed for the experiments are provided in Table IV.

We use CACTI [26] to estimate area and energy consumption of the different SRAM memories included in the accelerator. In addition, we have implemented in Verilog the rest of the components in the pipeline of the accelerator and we synthesized them using the Synopsys Design Compiler with a 28nm commercial library. The simulator provides the activity factors that are employed to estimate dynamic energy for the different components. We use the delay estimated by CACTI and the delay of the critical path reported by Design Compiler to set the target frequency so that the various hardware structures can operate in one cycle.

Regarding our datasets, we use the testset of audio files included in LibriSpeech [27] corpus, that consists of 5.4 hours of speech. On the other hand, we use the latest acoustic model for English language provided in Pocketsphinx. We also train several acoustic models using LibriSpeech train set to verify our proposed methods.

CPU	4x ARM Cortex A57 at 1.9 GHz (20 nm)
L1, L2	32KB L1, 2MB L2

Table II: Mobile CPU Parameters.

GPU	Tegra X1 Maxwell (GM204)
Streaming multiprocessors	256-core
Technology / Frequency	20 nm / 1000 MHz
L1, L2 caches	24KB [23], 256KB

Table III: Mobile GPU Parameters.

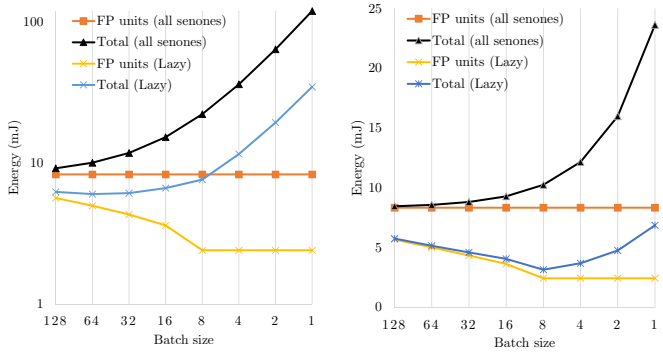
We compare our GMM accelerator with the performance of a software implementation running on a mobile CPU and a mobile GPU. We use NVIDIA Tegra X1 [6] as our baseline mobile platform. Tegra X1 includes an ARM CPU with parameters shown in Table II and a state-of-the-art mobile GPU with parameters provided in Table III.

Regarding the software implementation, we use the method for acoustic likelihood computation based on matrix-matrix multiplication described in [15]. In this method, the Gaussians and the input frames are represented as 2D matrices and the acoustic scores are obtained by using the matrix multiply (SGEMM) operation of BLAS specification. For the CPU version, we use SGEMM implementation available in the OpenBLAS library [1]. For the GPU version, we employ the high-performance implementation of SGEMM provided in cuBLAS [3]. To measure energy consumption, we read the registers of the TI INA3221 power monitor included in the NVIDIA Jetson TX1 platform, in order to obtain power dissipation by monitoring CPU and GPU power rails as described in [24].

VI. Experimental Results

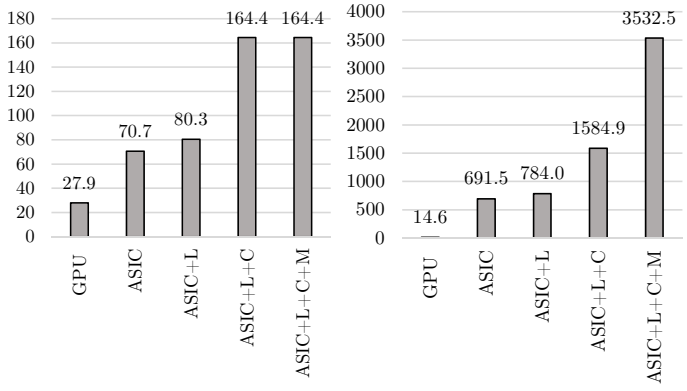
In this section, we evaluate the performance and energy consumption of our GMM accelerator presented in Section III. In first place, we analyze the impact of the batch size on the energy consumption of the accelerator. In second place, we compare our GMM accelerator with software solutions running on a mobile CPU and a mobile GPU.

Figure 12 shows the energy consumption of the FP units and the overall accelerator versus batch size. The figure shows the total energy for processing 128 frames of speech. For the base design of the accelerator that computes all the Gaussians every frame, labeled as *all senones*, the energy for the FP units is constant as the amount of computation does not depend on batch size. However, overall energy is significantly reduced when increasing batch size due to the energy required for memory transfers. The bigger the batch size the smaller the memory bandwidth usage, as Gaussians are fetched once and reused for a bigger number of frames, improving temporal locality. Energy consumption of base design (*all senones*) exhibits similar behavior using both the original dataset and the clustered dataset, but the absolute energy consumption is smaller for the clustered dataset (Figure 12b) due to the memory bandwidth reduction achieved with clustering.



(a) Original acoustic model (b) Clustered acoustic model

Figure 12: Energy consumption for floating point units (FP units) and entire GMM accelerator (Total), for the base design that evaluates all the Gaussians (all senones) and the accelerator using our lazy evaluation scheme (Lazy). 12a shows energy using the original uncompressed acoustic model, whereas 12b shows the results using our clustering scheme to reduce the size of the acoustic model.



(a) Speedup (b) Energy reduction

Figure 13: Speedup and energy reduction achieved by the mobile GPU and the different versions of the accelerator. The baseline is a modern ARM A57 mobile CPU.

Regarding our lazy evaluation scheme, the energy required for FP computations increases with the batch size. As described in Section III-B, the bigger the batch size the bigger the likelihood that a senone is active in at least one of the frames and, hence, the smaller the effectiveness in removing computations and memory accesses. For the original dataset (Figure 12a), overall energy is dominated by memory transfers and, hence, configurations with large batch sizes exhibit smaller energy consumption. However, for the clustered dataset (Figure 12b) the overall energy with the lazy scheme achieves the best results for a batch size of 8 frames. In this configuration, the memory bandwidth usage is reduced to a large extent due to the clustering, that reduces the size of the dataset by 8x, and the lazy scheme that avoids fetching GMM parameters for inactive senones.

On the other hand, Figure 12 also shows that both the lazy evaluation scheme and the clustering technique provide significant energy savings with respect to the base design for any batch size. Online ASR systems require small batch sizes of 4-16 frames to achieve high responsiveness, as buffering a large number of frames would introduce delays that are unacceptable for real-time systems. For small batch sizes, the configuration using lazy evaluation and clustering provides the lowest energy consumption, achieving the best results for a batch size of 8 as illustrated in Figure 12b. Therefore, we use 8 frames as the batch size of the accelerator for the rest of the results shown in this section.

Figure 13 shows the speedup and the energy reduction achieved by the GPU and the different versions of our GMM accelerator with respect to a modern mobile CPU. The configuration named *GPU* corresponds to the mobile GPU with parameters shown in Table III. *ASIC* corresponds to the base design of the GMM accelerator as described in Section III-A. *ASIC+L* is the base design including the lazy Gaussian evaluation scheme presented in Section III-B. *ASIC+L+C* is the GMM accelerator including lazy evaluation and the clustering technique introduced in Section IV-A. Finally, *ASIC+L-+C+M* configuration includes lazy evaluation, clustering and the memoization scheme described in Section IV-B.

The mobile GPU included in NVIDIA Tegra X1 provides 27.9x speedup and 14.6x energy savings with respect to the mobile CPU. The matrix multiply operation, that is used to implement the acoustic model in the software-based solutions, exhibits a high degree of data parallelism and, hence, it benefits from the large number of functional units provided by the GPU. In addition to the performance improvement, the GPU achieves high energy-efficiency for this data parallel code, reducing the overheads of instruction fetching and decoding by exploiting SIMD execution model.

On the other hand, the base design of the accelerator, named *ASIC* in Figure 13, provides 70.7x speedup and 691.5x energy reduction over the mobile CPU. The *ASIC* includes hardware specifically designed to accelerate GMM evaluation, avoiding the overheads of software implementations and delivering high-performance and energy-efficiency for acoustic scoring.

The lazy Gaussian evaluation scheme improves performance and energy-efficiency of the base design, achieving 80.3x speedup and 784x energy reduction over the mobile CPU. For a batch size of 8 frames, the lazy evaluation scheme avoids the computations and memory accesses for more than 40% of the Gaussians on average, as reported in Figure 7.

Our clustering technique provides large improvements in performance and energy consumption. The *ASIC+L+C* configuration achieves 164.4x speedup and 1584.9x energy reduction over the mobile CPU. We use our per-component clustering with 16 centroids for means and 16 for variances as described is Section IV-A. The per-component clustering provides 8x reduction in memory bandwidth usage, which

Platform	4x ARM A57	Tegra X1	B+L+C+M
Power (Watts)	2.45	9.22	0.11
Frames/s	95	2688	15802

Table IV: Performance and power of different processors.

results in speedups, since memory transfer time is one of the main bottlenecks for small batch sizes as reported in Figure 5, and energy savings as off-chip memory accesses are one of the most expensive operations in terms of energy for mobile SoCs.

Figure 13 shows that the memoization scheme does not provide any performance benefit, as this technique targets energy savings. Memoization does not increase the throughput of the accelerator, we still have 8 *Accumulators*. In the base design, the input of the *Accumulators* comes from the *Processing lanes*, whereas with the memoization scheme the input comes from the *Memoization Buffers* as illustrated in Figure 11. Hence, we replace the three FP operations performed in the *Processing lanes* by one access to the *Memoization Buffers*, that requires substantially less energy as described in Section IV-B. Overall, 74.88% of the FP operations are replaced by memory accesses to the memoization table, improving the energy reduction over the mobile CPU from 1584.9x in the *ASIC+L+C* to 3532.5x in the *ASIC+L+C+M*. Note that the memoization scheme requires a first stage for creating the memoization table. However, this precomputation only takes a negligible 0.15% of total execution time, as the number of entries in the memoization table, i. e. the number of possible combinations of means, variances and input features, is fairly small when using the clustered acoustic model.

Table IV shows power and performance of the different processors for GMM. The mobile CPU processes 92 frames/s while dissipating 2.45 W. The mobile GPU processes 2688 frames/s but at the cost of increasing power dissipation to 9.22 W. Our GMM accelerator achieves higher performance than the mobile GPU, processing 15802 frames/s while dissipating as low as 110 mW.

To sum up the energy-performance analysis, Figure 14 plots energy reduction vs speedup. The accelerator provides significantly higher energy-efficiency than the mobile processors. *ASIC+L+C+M* configuration achieves the best results, providing 5.89x speedup over the mobile GPU, while reducing energy consumption by 241x and requires an area of 0.94 mm^2. The system achieves 4.66x speedup and 4.54x energy reduction for the entire ASR pipeline.

VII. Related Work

Improving performance and reducing energy consumption of acoustic scoring has attracted the attention of the research community in recent years. Several hardware solutions for GMM evaluation have been proposed using ASICs or FPGAs. A hardware coprocessor is proposed in [22] to boost the performance of the GMM computation in Sphinx3. Reducing the size of the mantissa from 23-bits to 15-bits and

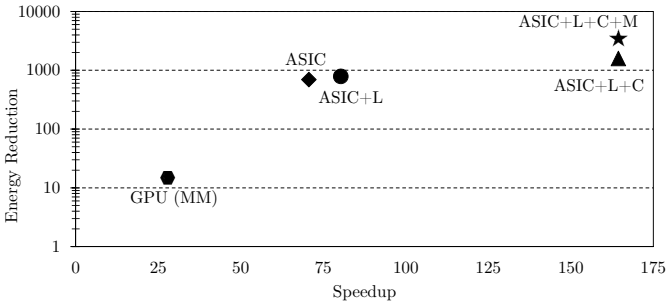


Figure 14: Speedup and energy reduction for different versions of the accelerator versus the mobile CPU and GPU.

12-bits is proposed in [11] to reduce the acoustic model size, providing a compression ratio of 1.39x and 1.6x respectively. The technique is evaluated using a small vocabulary size of 5000 words, whereas we propose a novel clustering technique to achieve 8x reduction in acoustic model size with a 130k words vocabulary size. A batching technique is proposed in [13], [20] to reduce the memory bandwidth. These works use small batch sizes of 4 and 8 in order to reduce the latency caused by batching. Our work is different as we combine batching with lazy evaluation by using a novel prediction scheme of active senones.

Our GMM accelerator is different from previous proposals as it includes lazy evaluation combined with batching, clustering and memoization, that provide significant performance and energy improvements in comparison with the basic design.

VIII. Conclusions

In this paper, we propose 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 and wearable devices. Our accelerator focuses on the evaluation of the Gaussian Mixture Model (GMM) for acoustic scoring, as this stage is the main bottleneck in speech recognition systems. Our design includes innovative techniques to improve its energy efficiency. First, it includes a lazy evaluation scheme and a prediction technique to compute Gaussian distributions on demand, avoiding 50% of the computations. Second, it employs a novel clustering technique to reduce memory bandwidth usage by 8x with a negligible impact on accuracy. Third, it implements a memoization scheme to remove redundant computations, avoiding 74.88% of the floating point operations. The final design achieves 5.89x speedup over a state-of-the-art software implementation running on a high-end mobile GPU, while reducing energy consumption by 241x.

Acknowledgment

This work was supported by the Spanish State Research Agency under grants TIN2013-44375-R and TIN2016-75344-R (AEI/FEDER, EU).

REFERENCES

[1] An optimized BLAS library (OpenBLAS). http://www.openblas.net/.

[2] Cortana. https://en.wikipedia.org/wiki/Cortana_%28software%29.

[3] cuBLAS. http://docs.nvidia.com/cuda/cublas/.

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

[5] Kaldi ASR results. https://github.com/kaldi-asr/kaldi/tree/master/egs.

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

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

[8] The Samsung Galaxy S7 Review. http://www.anandtech.com/show/10120/the-samsung-galaxy-s7-review.

[9] L. Bahl, R. Bakis, P. Cohen, A. Cole, F. Jelinek, B. Lewis, and R.L. Mercer. Further results on the recognition of a continuously read natural corpus. In *ICASSP'80.*, volume 5, pages 872–875, Apr 1980.

[10] Enrico Bocchieri. Vector quantization for the efficient computation of continuous density likelihoods. In *Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on*, volume 2, pages 692–695. IEEE, 1993.

[11] Dhruba Chandra, Ullas Pazhayaveetil, and Paul D Franzon. Architecture for low power large vocabulary speech recognition. In *2006 IEEE International SOC Conference*, pages 25–28. IEEE, 2006.

[12] Prakash Chockalingam. *Non-rigid multi-modal object tracking using Gaussian mixture models*. PhD thesis, Clemson University, 2009.

[13] Anthony Chun, Jenny X Chang, Zhen Fang, Ravishankar Iyer, and Michael Deisher. Isis: An accelerator for sphinx speech recognition. In *Application Specific Processors (SASP), 2011 IEEE 9th Symposium on*, pages 58–61. IEEE, 2011.

[14] Vassilios V Digalakis, Leonardo G Neumeyer, and Manolis Perakakis. Quantization of cepstral parameters for speech recognition over the world wide web. *IEEE Journal on selected areas in communications*, 17(1):82–90, 1999.

[15] Paul R Dixon, Tasuku Oonishi, and Sadaoki Furui. Harnessing graphics processors for the fast computation of acoustic likelihoods in speech recognition. *Computer Speech & Language*, 23(4):510–526, 2009.

[16] Rahman Farnoosh and Behnam Zarpak. Image segmentation using gaussian mixture model. *IUST International Journal of Engineering Science*, 19(1-2):29–32, 2008.

[17] Nicola Greggio, Alexandre Bernardino, Cecilia Laschi, Paolo Dario, and José Santos-Victor. Fast estimation of gaussian mixture models for image segmentation. *Machine Vision and Applications*, 23(4):773–789, 2012.

[18] Kshitij Gupta and John D Owens. Three-layer optimizations for fast gmm computations on gpu-like parallel processors. In *ASRU'2009*, pages 146–151. IEEE, 2009.

[19] 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.

[20] Ravi Iyer, Sadagopan Srinivasan, Omesh Tickoo, Zhen Fang, Ramesh Illikkal, Steven Zhang, Vineet Chadha, Paul Stillwell, and Seung Eun Lee. Cogniserve: Heterogeneous server architecture for large-scale recognition. *IEEE Micro, 31(3)*, 2011.

[21] Akinobu Lee and Tatsuya Kawahara. Recent development of open-source speech recognition engine julius. In *Proceedings of APSIPA ASC 2009*, pages 131–137, 2009.

[22] Binu Mathew, Al Davis, and Zhen Fang. A low-power accelerator for the sphinx 3 speech recognition system. In *CASES'03*, 2003.

[23] Xinxin Mei and Xiaowen Chu. Dissecting gpu memory hierarchy through microbenchmarking. 2015.

[24] Merlin Friesen. Linux Power Management Optimization on the Nvidia Jetson Platform. http://events.linuxfoundation.org/sites/events/files/slides/Linux_Low_Power_ELC_SanDiego.pdf.

[25] 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.

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

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

[28] 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*.

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

[30] Mosur Ravishankar, Roberto Bisiani, and Eric H Thayer. Subvector clustering to improve memory and speed performance of acoustic likelihood computation. In *EUROSPEECH*, 1997.

[31] Zs Robotka and A Zempléni. Image retrieval using gaussian mixture models. *Annals Univ. Sci. Budapest, Sect. Comp*, 31:93–105, 2009.

[32] 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.

[33] 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.

[34] S Young, G Evermann, M Gales, T Hain, D Kershaw, X Liu, G Moore, J Odell, D Ollason, D Povey, et al. The htk book (v3. 4). *Cambridge University*, 2006.