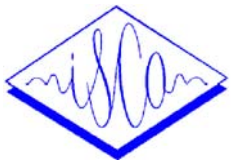

INTERSPEECH 2004 - ICSLP
8th International Conference on Spoken Language Processing
ICC Jeju, Jeju Island, Korea
October 4-8, 2004

Towards large vocabulary ASR on embedded platforms

Miroslav Novak

IBM T.J. Watson Research Center
P. O. Box 218, Yorktown Heights, NY 10598, USA
miroslav@us.ibm.com

Abstract

In this paper we present an overview of an automatic speech recognition system implementation in the context of embedded systems. Specific challenges presented by low resource platforms will be addressed for the basic components of an ASR decoder. Our main objective is to utilize and modify the technology developed for large vocabulary ASR to achieve efficient LVCSR on embedded systems as well.

1. Introduction

ASR for embedded systems is becoming increasingly popular. The targeted platforms include mobile devices such as mobile phones, PDAs as well as low-cost solutions for multi-modal interfaces in cars etc. As the popularity of ASR grows, we can expect increasing demand for functionality. For example, replacing simple command and control grammar based applications by natural language understanding (NLU) systems leads to an increased vocabulary sizes. While systems for real-time large vocabulary ASR have been available for many years, deployment on platforms with limited resources presents new challenges in many aspects of the overall ASR design [1].

Embedded ASR can be deployed either locally or in a distributed environment. The advantages of distributed ASR are clear as the device only performs waveform capture and feature computations and the bulk of the computational cost is moved to a server [2]. Recent efforts in feature standardization have contributed to the popularity of this approach. Since the ASR server availability cannot be guaranteed at all time, distributed systems are not well suited for interactive applications. Local ASR significantly reduces the required bandwidth in transactional systems or eliminates the need completely if access to a remote database is not needed. In addition, user customization of local ASR systems is easier, since all changes (e.g. acoustic model adaptation, vocabulary customization) do not need to be transfered and preserved on the server side.

In this paper we present an overview of our LVCSR decoder design from a limited resource implementation point of view. A block diagram of the system is shown in figure 1. We will focus on those parts particularly affected and we will present our approaches to particular problems and compare them to other solutions published in the literature.

The rest of the paper is organized as follows: Section 2 describes the main problems encountered on embedded platforms. In section 3, we discuss techniques used for fast Gaussian evaluation. We look at search implementation in section 4, and search graph compilation in section 5. Use of fast match is mentioned in section 6 and finally, in section 7, we briefly discuss alternative search methods.

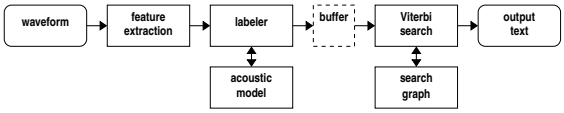

Figure 1: Organization of the decoder

2. Limitations of embedded platforms

In comparison to a typical workstation used for large vocabulary ASR, the embedded platforms are limited in terms of CPU power and amount of memory. While the workstation market is dominated by a single platform (i.e. Intel), the situation in embedded devices is much more diverse (both 16 and 32 bit architectures are common), which makes the development of one-size-fits-all implementation very difficult if not impossible.

Memory, not only the amount available but, more importantly, its speed seems to be the most limiting factor. It is well known that a fast CPU is not very helpful when it is used in combination with slow memory, especially when a very small memory cache is used.

There is a popular belief that in implementation of many algorithms, speed can be improved by using more memory, and, vice versa. Caution is needed with embedded systems, because the use of more memory may lead to a higher cache miss rate and a degradation in performance. Algorithmic improvements developed on a large computer do not always translate to improvements on systems with limited resources.

Most embedded platforms have only several kilobytes of data cache and very small Translation Lookaside Buffers (e.g. 32 entries). This limitation has a severe impact on algorithms which need a large number of memory pages - TLB faults can cost hundreds of machine cycles.

Typical embedded CPUs also often lack hardware support for floating point operations, so many algorithms need to be redesigned to work efficiently in integer arithmetic.

3. Front end

For the ASR front-end, we consider those processes which take the waveform as input and produce state observation likelihoods. The computational cost of feature extraction (cepstral coefficients) is relatively small in comparison to the rest of the decoder. However, the cost of the observation likelihood computation is significant.

The likelihoods are almost exclusively based on Gaussian mixtures of some form (untied, tied etc.). Regardless of the particular scheme, the number of Gaussians needed for high accuracy will likely be much greater than the number Gaussians which can be evaluated at every time frame. It is clear that

number of Gaussians	ranks	mixtures
1000	8.51	9.31
1500	8.25	8.43
2000	8.19	8.10
3000	8.10	8.05

Table 1: String error rate comparison of rank and mixture likelihood based systems

not all Gaussians are always needed. There exist two basic approaches to reducing Gaussian evaluation: on demand computation and Gaussian selection. In the on demand scheme only those mixtures corresponding to the active state are computed. This approach does not scale well; as the search space grows, the number of active mixtures grows as well. As the number of components in a mixture grows, the on demand computation is even less effective, since only very few components contribute significantly to the mixture likelihood (we have verified experimentally that the use of the best component only is usually sufficient). Another disadvantage is that the likelihoods are computed synchronously with the search.

In selection based schemes, the goal is to evaluate only a limited number of Gaussian components, either completely independently of their states or in some other manner which will achieve reasonable state coverage. Popular techniques are those which create a shortlist of components; some form of an inexpensive metric is used to partition the feature space and to create an active shortlist(s) given the feature vector [3],[4]. Our preferred method [5] uses a n-ary search tree with a non-overlapping shortlist. These non-overlapping lists can be represented much more efficiently in memory. At each level of the tree, one Gaussian is used to represent each shortlist at the next level. Quantization is used to reduce the memory storage for both means and variances.

In the selection scheme, the state likelihood computation can be completely decoupled from the search. It can be performed on blocks of feature vectors rather then synchronously. This can significantly improve the memory cache utilization.

Rather than using the mixture likelihood directly, the IBM system uses the probability of the rank of the mixture [6]. The output distribution represents the probability that a given mixture will have a certain rank when sorted by likelihood. The concept of rank distribution was originally introduced in our asynchronous stack decoder implementation and its main purpose was to normalize the likelihood range. We have found that its use has advantages in Viterbi decoder as well. A likelihood value from the tail of the rank distribution provides a robust estimate for states which are not evaluated in the Gaussian selection scheme. We have observed that the rank based likelihoods are more robust against underestimation when too few Gaussians are evaluated. Table 1 shows comparison of string error rates for rank and mixture based systems when the number of components is limited in a simulated experiment (i.e. the top Gaussians were the best matching ones). It can be seen that although, eventually, the mixtures are slightly better, the ranks achieve good accuracy with fewer Gaussians evaluated. In our real Gaussian selection setup, where the evaluated Gaussians are not guaranteed to be the best ones, the advantage of ranks is even more apparent.

The use of a rank distribution also has an effect on the state transition probabilities. While many believe that transition probabilities do not play a significant role due to the huge dynamic range of mixture likelihoods in comparison to the range of transition probabilities, the rank distribution brings the dynamic range of the observation likelihoods closer to the one of transition probabilities. We have observed that the use of transition probabilities can improve the accuracy, in particular when the language model is weak, and in a presence of noise.

Computational cost is only one factor affecting the choice of acoustic model complexity. For example the choice of the context modeling scope has an impact on the decoding graph construction. Since the use of word-internal context model simplifies this construction significantly, it has been our model of choice for embedded platforms. The loss of accuracy we have observed on our test set is about 5%, but it can be higher for some tasks, on random length digits strings it is more than 9%. Since the digit task is quite essential, we improve its accuracy by using digit specific phones.

4. Search

In LVCSR, search is clearly the most computationally expensive part of the decoder. An excellent overview of search techniques can be found in [7]. In ViaVoice, the IBM large vocabulary ASR system, an asynchronous stack based decoder is used. In combination with an efficient fast match, the stack decoding scheme is very memory efficient. But the complexity of the algorithm makes it less suitable for embedded platforms. Instead, the Embedded ViaVoice [8] uses a Viterbi decoder.

Recent comparisons [9], [10] show that the use of transducers to produce a minimized static graph is the most computationally efficient design for Viterbi based decoders on unrestricted platforms. Embedded system memory restrictions put a limit on the size of the usable acoustic and language models. Large models may require a dynamic scheme (e.g. [11]).

The details of a Viterbi search implementation can vary significantly. It is difficult to objectively compare the various techniques found in the literature; the speed of the decoder results from mutual interaction of many factors, such as type and speed of CPU, compilers, amount and speed of memory, task complexity, etc.

We will focus our comparison mainly on memory organization. Memory usage in a Viterbi decoder can be divided into three classes:

- Static representation of the HMM states - *graph space*,
- memory for active states, updated at each time frame - *search space*,
- traceback information (tokens).

All information related to the graph space can be possibly stored in read-only memory, while both the search space and traceback inherently require read-write memory to evaluate likelihoods for each frame and to propagate the traceback information. The search can be implemented in three distinct ways:

Combination of the static graph and static search space represents the most efficient representation from the search point of view. Here all memory is allocated before the search starts with one to one correspondence between the graph state and search state. This arrangement minimizes overhead during the search. The static search space can represented in a very compact form, using local properties of the network, e.g. linear sequences of HMM states can be stored very efficiently without need for explicit connections between states.

This method is very efficient for small tasks, but does not scale well to large systems with large vocabularies. As the size of the network grows, a large portion of the search space memory is wasted since only a small portion of it will be used. It is

difficult to take advantage of more complex memory reduction techniques such as graph factoring [12].

Static graph space with dynamic search space is a more memory efficient option. The amount of search space memory corresponds to the number of active states at each frame. This also gives us the option to limit the amount of search space memory by using pruning techniques. There is some run time overhead associated with the mapping between the graph and search spaces, which changes for each time frame. On the other hand the improved locality of the memory access due to a much smaller search space can significantly improve the search speed. Typically the more complicated the search graph representation, the more expensive the search space construction becomes.

Graph factoring can be used to reduce the memory needed for the search graph, but for the most efficient search space implementation (i.e. the dynamic assignment of the search memory to the corresponding states in the graph space) the graph representation needs to be simple, i.e. without factoring. The efficiency of this approach has been demonstrated on LVCSR in the context of the Switchboard 1xRT evaluation [13].

Dynamically built graph offers the most memory efficient decoder, but there can be a significant cost associated with the graph building. The advantage is obvious - only the part of the graph which is searched is constructed in memory, with possibly significant memory savings in comparison to static graph methods. The savings depends on the nature of the task, i.e. on the ratio between the expected number of states visited during the search and the static graph size. This is the technique of choice when the static graph size is too large or when it is not practical to build the full graph at all. On systems with limited memory, this approach clearly offers benefits, but the dynamic graph building algorithm needs to be carefully designed to limit its overhead cost.

Techniques which try to combine advantages of the static and dynamic methods have been proposed, in [14] the part of the graph corresponding to the unigram language model is statically compiled, and [15] proposes an incremental application of a factorized language model to the search graph.

5. Search graph construction

In comparisons of the static graph scenario with dynamic decoders the time needed to construct the graph is usually not included in the real-time factor computation. But the assumption that the search graph never changes is clearly not practical. The complexity of the graph construction algorithm plays an important role as well. The ability to construct a search efficiently is desirable for several reasons. Even if the static graph scenario is used, there may exist a need to quickly compile the graph using the limited resources of the particular platform before the search starts, for example when the grammar is constructed dynamically in response to a certain dialog state. Ability to customize the system by the user also requires that the graph be built locally.

The use of finite state transducers has become popular in the speech community [12]. It provides a solid theoretical framework for the operations needed to create a search graph. The search graph is the result of a composition

$$C \circ L \circ G \tag{1}$$

where G represents the language model as a finite state acceptor (FSA), L represents the pronunciation model and C converts the context independent phones to context dependent HMMs.

Language models fall into two distinct categories: n-grams and grammars. Each has different properties from the graph construction point of view.

For n-gram models, the back-off model type is widely preferred for use in a Viterbi decoder. A straightforward approximate method can be used to construct the FSA representation of an n-gram back-off model using non-emitting (null) arcs representing the back-off transitions. If the back-off symbol is treated as a part of the vocabulary, G can be considered deterministic. It would be impractical, and even impossible, to determinize the n-gram model by removing the null arcs. Each state represents a unique history and the graph is usually close to being minimal.

Grammars represent a way to define a set of (possibly infinite) of allowed sentences. Most of the systems use a formal syntax based on Context Free Grammars with regular expression construct extensions to enable more compact representation. By allowing only right recursion, regular languages can be generated and a corresponding FSA can be found.

The task of a grammar compiler is to convert the grammar definition (e.g. written in BNF language) into a (weighted) FSA. The first step of the compiler usually produces a nondeterministic WFSA. The next step is determinization, usually the most expensive part of the process (possibly with exponential cost). In some situations determinization can significantly increase the number of states and arcs. The final compilation step finds a minimal form of the deterministic WFSA. The complexity of minimization is $O(NA\log(N))$ where N is number of states and A is average branching factor. For acyclic FSA much faster ($O(N + A)$) minimization algorithms exist.

Various schemes have been proposed for more efficient search graph compilation of grammars. A common idea is to avoid full expansion of the search graph and compile only the part used at search time by exploiting properties of that particular grammar.

Concept of Recursive Transition Networks and a late binding scheme is used in [16], and [17] proposes the use of hierarchical non-deterministic grammar compilation. While we believe this is an important research topic, the scope of this paper does not allow us to explore this topic in more detail. For the rest of the paper we will assume an optimized G has been provided.

To construct the search graph, we use a technique which is less general but allows us to achieve high memory efficiency at low computational cost. Both determinization and minimization are simpler when performed on finite state acceptors. By not using the transducer framework, we do not take full advantage of potential minimization by pushing the output labels. However, placement of word labels at the ends of the words simplifies the decoder design as well (proper time alignment is preserved). We have found out that, in practice, full minimization (with label pushing) generally reduced the graph very little in most cases.

The method we use to compile a search graph performs all operations (composition, determinization, minimization and pushing) in a single step, applied incrementally to each state of G (and for each context for cross-word context models) one at a time [18]. The resulting graph is fully minimized (within the FSA framework) with no intermediate step requiring more memory than the final graph. In other words, during the compilation, the graph never has more arcs or states than the final one. An important advantage of the incremental construction is that it does not require use of a general minimization procedure, rather a very efficient local acyclic graph minimization is used.

6. Fast match

Search cost can be reduced by the use of a fast match. The idea is to use inexpensive models to look into the near future to decide which paths can be pruned. The effectivnes of the fast match increases with the accuracy of the approximate models and the amount of look ahead time and decreases with the computation costs, so a tradeoff needs to be found. In the asynchronous stack decoder we use fast match very efficiently with a look-ahead of whole words. In a synchronous scheme, that is difficult to do, so the popular schemes (e.g. [19]) are usually limited to prediction of the next active phone. In this scope, most of the time savings comes from elimination of the observation likelihood for a state. In the Gaussian selection scheme where the Gaussian computation is independent of the search, the short term fast match is less effective.

We have been able to use long-term fast match for certain tasks [20]. When the utterances are relatively short and the grammar can be effectively expressed as a tree, then we can use a two pass method with inexpensive phonetic models used in the first pass to find top N best paths to be rescored in a second pass.

7. Alternative decoding schemes

A decoding strategy based on building HMM networks (either statically or dynamically) has limitations. Memory use is certainly an issue for static graphs, but even for dynamically build graphs, the expansion of the active space may be too memory intensive. The combination of the acoustic and the language model in the search graph may not be the most optimal approach in all situations. The context affecting the acoustic search (several phones) is typically much shorter than the context affecting the language model (several words). The acoustic search space can be much smaller without (or with significantly reduced) language model constraints.

This idea is utilized in the multi-pass decoding schemes, where a more complex model is applied at each pass. The inherent latency of multi-pass decoding makes it less attractive for interactive systems.

The time conditioned search [21] can be seen as a method which performs acoustic search independently of the language model in a single pass decoding scheme. A significant advantage was not found when 3-gram model was used, as the cost of recombination with the language model reduced the efficiency of the method. One can expect that for much more complex models the benefits would be more apparent.

In a different scheme, [22] uses synchronous acoustic search and asynchronous search for the language model part.

In an asynchronous decoder, the AM and LM decoupling is used [23] to achieve a significant speed improvement since the recombination cost is very small (using some approximations not affecting the accuracy).

8. Conclusion

We have presented a variety of design choices for the basic components of an LVCSR system. Proper combination of the key choices is essential in order to achieve deployment of a LV system on platforms with limited resources. We tried to point out those research areas which could potentially lead to significant improvements in the search efficiency.

9. Acknowledgments

The author acknowledges the contributions of members of the embedded ASR team lead by Satya Dharanipragada and at IBM Voice Technologies and Systems group lead by Jan Sedivy in Prague. The author also benefited from useful discussions with George Saon, Stanley Chen and Geoffrey Zweig.

10. References

[1] O. Viikki, "ASR in portable wireless devices," in *Proc. of ASRU '01*, 2001.

[2] R.C. Rose, S. Parthasarathy, B. Gajic, A.E. Rosenberg, and S. Narayanan, "On the implementation of ASR algorithms for hand-held wireless mobile devices," in *Proceedings of ICASSP '01*, 2001, vol. 1, pp. 17–20.

[3] M.J.F. Gales, K.M. Knill, and S.J. Young, "State-based gaussian selection in large vocabulary continuous speech recognition using hmms," *IEEE Transactions on Speech and Audio Processing*, vol. 7, no. 2, pp. 152 – 161, 1999.

[4] S. Ortmanns, T. Firzlaff, and H. Ney, "Fast likelihood computation for continuous mixture densities in large vocabulary speech recognition," in *Proc. of Eurospeech '97*, 2002, pp. 143–146.

[5] M. Novak, R. A. Gopinath, and J. Sedivy, "Effi cient hierarchical labeler algorithm for gaussian likelihoods computation in resource constrained speech recognition systems," *http://www.research.ibm.com/people/r/rameshg/novak-icassp2002.ps*.

[6] L.R. Bahl P.V. de Souza, P.S. Gopalakrishnan, D. Nahamoo, and M.A. Picheny, "Robust methods for using context-dependent features and speech recognition models in a continuous speech recognizer," in *Proc. ICASSP '94*, 1994.

[7] X.L. Aubert, "An overview of decoding techniques for large vocabulary continuous speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 89–114, January 2002.

[8] S. Deligne et al., "Robust high accuracy speech recognition system for mobile applications," *IEEE Trans. Speech and Audio Proc.*, vol. 10, no. 8, pp. 551–561, 2002.

[9] S. Kanthak, H. Ney, M. Riley, and M. Mohri, "A comparison two lvr search optimization techniques," in *Proc. of ICSLP '02*, 2002, pp. 1309–1312.

[10] H. Dolfi ng, "A comparison of prefi x tree and fi nite-state transducer search space modelings for large vocabulary speech recognition," in *Proc. of ICSLP '02*, 2002, pp. 1305–1308.

[11] S. Ortmanns and A. Eiden H. Ney, "Improved lexical tree search for large vocabulary speech recognition," in *Proceedings of ICASSP '98*, 1998, vol. 2, pp. 817–820.

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

[13] G. Saon, G. Zweig, B. Kingsbury, L. Mangu, and U. Chaudhari, "An architecture for rapid decoding of large vocabulary conversational speech," in *Proc. of Eurospeech '03*, 2003, pp. 1977–1980.

[14] D. Willet and S. Katagiri, "Recent advances in effi cient decoding combining on-line transducer composition and smoothed language model incorporation," in *Proc. of ICASSP '02*, 2002, pp. 713–716.

[15] H.J.G.A. Dolfi ng and I.L. Hetherington, "Incremental language models for speech recognition using fi nite-state transducers," in *Proc. of ASRU '01*, 2001, pp. 194–197.

[16] J. Schalkwyk, L. Hetherington, and E. Story, "Speech recognition with dynamic grammars using fi nite-state transducers," in *Proc. of Eurospeech '03*, 2003, pp. 1969–1972.

[17] J. Zheng and H. Franco, "Fast hierarchical grammar optimization algorithm toward time and space effi ciency," in *Proc. of ICSLP '02*, 2002, pp. 393–396.

[18] M. Novak and V. Bergl, "Memory effi cient decoding graph compilation with wide cross-word accoustic context," in *Submitted to ICSLP 2004*.

[19] S. Ortmanns, H. Ney, A. Eiden, and N. Coenen, "Look ahead techniques for fast beam search," in *Proceedings of ICASSP97*, 1997, pp. 1783–1786.

[20] M. Novak, R. Hampl, P. Krbec, V. Bergl, and J. Sedivy, "Two-pass search strategy for large list recognition on embedded speech recognition platforms," in *Proceedings of ICASSP '03*, 2003, vol. 1, pp. 200–203.

[21] S. Ortmanns and H. Ney, "The time-conditioned approach in dynamic programming search for LVCSR," *IEEE Transactions on Speech and Audio Processing*, vol. 8, no. 6, pp. 676 – 687, 2000.

[22] S. Renals and M.M. Hochberg, "Start-synchronous search for large vocabulary continuous speech recognition," *IEEE Transactions on Speech and Audio Processing*, vol. 7, no. 5, pp. 542 – 553, 1999.

[23] M. Novak and M. Picheny, "Speed improvement of the tree-based time asynchronous search," in *Proc. ICSLP 2000*, Beijing, November 2000, pp. 334–337.