

Efficient Manycore CHMM Speech Recognition for Audiovisual and Multistream Data

Dorothea Kolossa^{1,2}, Jike Chong¹, Steffen Zeiler², Kurt Keutzer¹

¹Department of Electrical Engineering and Computer Science, University of California, Berkeley

²Fachgebiet Elektronik und medizinische Signalverarbeitung, Technische Universität Berlin

d.kolossa@ee.tu-berlin.de, jike@chongjike.net, steffen.zeiler@gmx.de, keutzer@eecs.berkeley.edu

Abstract

Robustness of speech recognition can be significantly improved by multi-stream and especially by audiovisual speech recognition. This is of interest for example for human-machine interaction in noisy reverberant environments, and for transcription of or search in multimedia data. The most robust implementations of audiovisual speech recognition often utilize Coupled Hidden Markov Models (CHMMs), which allow for both modalities to be asynchronous to a certain degree. In contrast to conventional speech recognition, this increases the search space significantly, so current implementations of CHMM systems are often not real-time capable.

Thus, for real-time constrained applications such as online transcription of VoIP communication or responsive multi-modal human-machine interaction, using current multiprocessor computing capability is vital. This paper describes how general purpose graphics processors can be used to obtain a real-time implementation of audiovisual and multi-stream speech recognition. The design has been integrated both with a WFST-decoder and a token passing system, with parallelization leading to a maximum speedup factor of 32 and 25, respectively.

Index Terms: audiovisual speech recognition, multistream speech recognition, coupled HMM, GPU

1. Introduction

Multistream and audiovisual speech recognition both use a number of streams of audio and/or video features in order significantly increase robustness and performance ([1, 2]). Coupled hidden markov models (CHMMs), with their tolerance for stream asynchronicities, can provide a flexible integration of these streams and have shown optimum performance in a direct comparison of alternative model structures in [3].

In CHMMs, both feature vector sequences are retained as separate streams. As generative models, CHMMs describe the probability of both feature streams jointly as a function of a set of two discrete, hidden state variables, which evolve analogously to the single state variable of a conventional HMM.

Thus, CHMMs have a two-dimensional state \mathbf{q} which is composed of an audio and a video state, q_a and q_v , respectively, as shown in Fig. 1. Each possible sequence of states through the model represents one possible alignment with the sequence of observation vectors. To evaluate the likelihood of such an alignment, each state pairing is connected by a transition probability, and each state is associated with an observation probability distribution.

The transition probability and the observation probability can both be composed from the two marginal HMMs. Then,

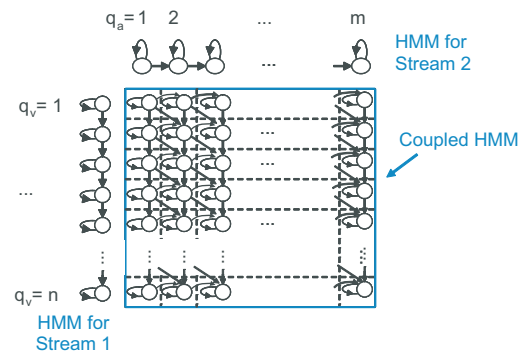


Figure 1: A coupled HMM consists of a matrix of interconnected states, which each correspond to the pairing of one audio- and one video-HMM-state, q_a and q_v , respectively.

the coupled transition probability becomes

$$p(q_a(t+1) = j_a, q_v(t+1) = j_v | q_a(t) = i_a, q_v(t) = i_v) = a_a(i_a, j_a) \cdot a_v(i_v, j_v) \quad (1)$$

where $a_a(i_a, j_a)$ and $a_v(i_v, j_v)$ correspond to the transition probabilities of the two marginal HMMs, the audio-only and the video-only single-stream HMMs, respectively. This step of the computation, the so-called *propagation step*, is memory intensive due to the need for transition probability lookup in a large and irregularly structured network.

For the observation probability, both marginal HMMs could equally be composed to form a joint output probability by

$$p(\mathbf{o}|\mathbf{i}) = b_a(o_a|i_a) \cdot b_v(o_v|i_v). \quad (2)$$

Here $b_a(o_a|i_a)$ and $b_v(o_v|i_v)$ denote the output probability distributions for both single streams.

However, such a formulation does not take into account the different reliabilities of audio and video stream. Therefore, Eq. (2) is commonly modified by an additional stream weight γ as follows

$$p(\mathbf{o}|\mathbf{i}) = b_a(o_a|i_a)^\gamma \cdot b_v(o_v|i_v)^{1-\gamma}. \quad (3)$$

Finally, computation of the marginal HMM state probabilities can be implemented as, e.g., an M-component Gaussian mixture model (GMM)

$$b(o|i) = \sum_{m=1}^M \gamma_m \mathcal{N}(o | \mu_{i,m}, \Sigma_{i,m}). \quad (4)$$

Here, $\mathcal{N}(o|\mu, \Sigma)$ stands for a multivariate Gaussian distribution evaluated for the vector-valued observation o with mean μ and

covariance matrix Σ . The covariance matrix may be either a full or a diagonal covariance matrix, where the latter assumes implicitly that feature vector components are independent or that their dependences may be neglected. In addition to these standard Gaussian likelihood functions, we have additionally implemented robust likelihood kernels which are capable of taking into account the time-varying reliabilities of all feature components. Such strategies have recently been shown successful for audiovisual speech recognition [4, 5] and are also of interest here, due to their increased computational complexity. Therefore, one exemplary robust likelihood function, *modified imputation* [6], will also be considered in the following.

The steps given by (4) and (3) will be referred to as the *likelihood computation* and the *likelihood combination* steps, respectively. These steps, especially the *computation*, have a much greater compute-to-memory-access ratio than the propagation step, due to the computational effort involved in GMM evaluation.

2. Prior Work

A number of recent publications have focussed on efficient parallelization of large-vocabulary HMM-based speech recognition using current multi-core and manycore processors [7, 8, 9]. While [9] and [7] utilized the GPU for likelihood computation only, current work in [8] describes a system that also performs the token propagation on the GPU. However, to our best knowledge, no parallel implementations of audiovisual speech recognition or coupled HMM decoding have been presented so far.

3. Parallelization of Coupled HMM Decoding

In order to efficiently parallelize CHMM decoding, it is vital to consider all sources of parallelism that are inherent in the task.

However, the breadth of considered likelihood functions, which may be full- or diagonal-covariance models with or without modified imputation, with widely varying numbers of mixture components and of feature vector dimensions, only a subset of all potential parallelism sources remains to be exploited.

The sources of parallelism that are applicable for all considered model types are

- parallelism over time frames
- over marginal or coupled CHMM state
- and over the two algorithm phases, namely the likelihood computation and combination steps; and the propagation step.

The presented strategy utilizes all above parallelism sources, with state parallelism only utilized across the marginal but not across the coupled states. In that respect, the overall system design is closely related to the implementation described in [7], where likelihood functions are also evaluated on the GPU whereas the search takes place on the CPU side in parallel.

3.1. Likelihood kernels

To adapt the likelihood calculations to the idiosyncracies of the GPU hardware, we transform the equation for the multivariate Gaussian

$$\mathcal{N}(x | \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \exp\left(-\frac{1}{2}(x - \mu)^\top \Sigma^{-1} (x - \mu)\right)$$

into a more suitable form for SIMD processing. Given the dynamic range, log-likelihoods are a necessity for floating point

calculations

$$\log(\mathcal{N}) = -\log\left(\sqrt{(2\pi)^D |\Sigma|}\right) - \frac{1}{2}(x - \mu)^\top \Sigma^{-1} (x - \mu)$$

and expressed as a quadratic form

$$\log(\mathcal{N}) = s + u^\top x - (Vx)^\top (Vx) \quad (5)$$

this leads to an algorithm that can be implemented as a single stream of multiply and sum operations without any branches and loops. For the calculation of the required coefficients s , u and V , sorting terms gives

$$\begin{aligned} \log(\mathcal{N}) &= -\frac{1}{2}\left(\log(2\pi)D + \log(|\Sigma|)\right) \\ &\quad - \frac{1}{2}\left(x^\top \Sigma^{-1} x - 2\mu^\top \Sigma^{-1} x + \mu^\top \Sigma^{-1} \mu\right) \\ &= -\frac{1}{2}\left(\log(2\pi)D + \log(|\Sigma|) + \mu^\top \Sigma^{-1} \mu\right) \\ &\quad + \mu^\top \Sigma^{-1} x - \frac{1}{2}x^\top \Sigma^{-1} x. \end{aligned} \quad (6)$$

Because any positive definite symmetric matrix A can be factored into a product of upper triangular matrices U

$$A = U^\top U$$

by means of the Cholesky decomposition, one can substitute the inverse of the covariance matrix Σ^{-1} with $U^\top U$.

$$x^\top \Sigma^{-1} x = x^\top U^\top U x = (Ux)^\top (Ux)$$

For the sake of a compact notation, this is written as the matrix square root $U = \Sigma^{-\frac{1}{2}}$. Inserted into (6), this yields

$$\begin{aligned} \log(\mathcal{N}) &= -\frac{1}{2}\left(\log(2\pi)D + \log(|\Sigma|) + \mu^\top \Sigma^{-1} \mu\right) \\ &\quad + \mu^\top \Sigma^{-1} x - \frac{1}{2}\left(\Sigma^{-\frac{1}{2}} x\right)^\top \left(\Sigma^{-\frac{1}{2}} x\right). \end{aligned}$$

After equating coefficients with (5), this gives the required coefficients

$$\begin{aligned} s &= -\frac{\log(2\pi)D + \log(|\Sigma|) + \mu^\top \Sigma^{-1} \mu}{2}, \\ u &= \mu^\top \Sigma^{-1}, \quad V = \frac{\Sigma^{-1/2}}{\sqrt{2}}. \end{aligned}$$

In case of multivariate Gaussian density functions using a full covariance matrix, because of symmetries only the $D(D+1)/2$ entries in the upper triangular part of Σ are used. For diagonal covariance matrices, a vector with D diagonal entries is stored. All GPU log-likelihood kernel functions for densities with diagonal covariances use a simplified version of the above algorithm, with less calculations and easier covariance matrix inversion. The missing feature log-likelihood kernel functions for modified imputation use diagonal covariance matrices as well in a straightforward manner.

3.2. Combination of Marginal to Coupled Likelihood

The computation of the coupled log-likelihoods from the marginals of audio and video observations according to Equation (3) is carried out only for active states (states that are currently carrying a token). This saves time but causes a very irregular data access pattern. This step is therefore implemented on the CPU (e.g. as Phase 1b in Fig. 2) and relies on the GPU calculations of Gaussian mixture models.

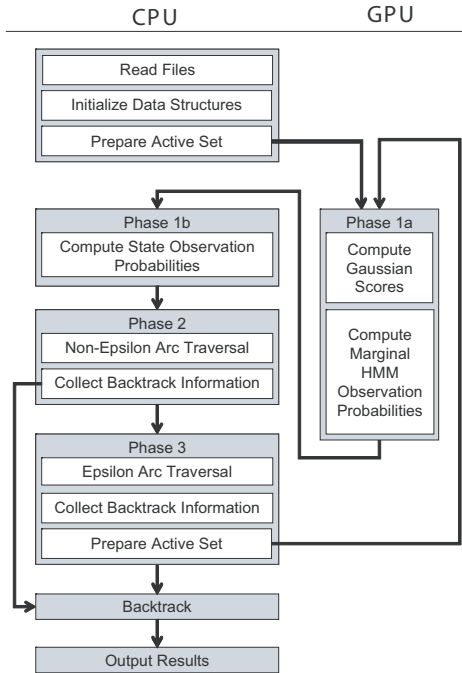


Figure 2: Block diagram of WFST system.

3.3. State Machine

The log-likelihood kernel functions on the GPU require the data in blocks of 32 consecutive time frames and should run in parallel to the CPU code. To handle these requirements, a state machine on the CPU keeps track of the starting point for the next block, the offset into the current block, and the appropriate function calls to calculate the log-likelihoods for Gaussian densities and GMMs. The inference engine on the CPU needs a set of log-likelihoods for every time frame, so the data of the current block is buffered while the computation for the next block is running on the GPU. Every 32nd time frame, the results from the previous block are copied back to the CPU, the block-offset is reset and the parallel computations on the GPU for the next block are initiated. Data sequences are not guaranteed to have a length that is a multiple of the block size. Therefore, the pointers of the last block are adjusted, such that the end of the last block is aligned with the end of the sequence and the computation of a complete data block is possible.

4. Experiment Setup

4.1. Architecture of C++-based WFST Decoder

Figure 2 shows the data flow of the weighted finite state transducer (WFST) decoder, which is described in detail in [8]. Coupled HMM models for this system were compiled and minimized from the marginal HMMs and the GRID sentence grammar [10] by using OpenFST ([11]). The compiled and minimized network comprised 3167 states and 12751 arcs.

4.2. Architecture of JAVA-based token passing system

The JAVA decoder *JASPER* uses a token passing architecture as described in [5]. The token passing scheme allows a separation of the inference from the underlying pattern matching technology [12]. Observation log-likelihood computations can be carried out easily in parallel on the GPU (similar to Phase 1a in Fig. 2), while likelihood results are accessed via JNI (Java Native Interface) function calls to the state machine, and can be used to update scores of the tokens passed around the network on the CPU.

For this purpose, a transition network is constructed from a grammar or language model, and each vocabulary element in the network is represented by a CHMM. Transitions between neighboring states have associated transition costs. Tokens are propagated along these transitions in a time synchronous manner and their scores are updated with transition and observation costs. Only the best scored tokens in every state survive and are propagated to the next time step. With a global threshold, the process is equivalent to a Viterbi beam search among partial paths through the word lattice for a given observation sequence.

4.3. Development System

All experiments were performed on a standard PC running Windows XP. All kernels and the C++ WFST decoder were compiled using Visual Studio 2008 with `nvcc` 2.2.

The CPU was an Intel Core i7 920 with 2.7GB of RAM and a clock frequency of 2.67 GHz and for GPU calculations, an Nvidia GTX 285 with 2GB of dedicated memory was used. The GTX285 has 30 cores with 8-way vector arithmetic units and is clocked at 1.51GHz.

5. Results

We have measured the accuracy and speedup for both systems, the JAVA-based one with token passing and the C++-based WFST recognizer. All speedup figures are given as a speedup factor

$$S = \frac{T_1}{T_p}, \quad (7)$$

where T_1 is the execution time of the sequential algorithm and T_p denotes the runtime when CPU and GPU are used in parallel as described in Section 3.

Results are shown in Tab. 1 and Fig. 3 for two different experiments, on the one hand the audiovisual recognizer in JAVA, and on the other hand the multi-stream models implemented in C++. In both cases, the accuracy consistently remained the same in the GPU implementation when compared to the reference CPU implementation. The accuracy is also the same over all numbers of mixture components, since it was our aim to show the structural speedup which occurs when all beamwidths and best paths remain equal. These accuracy figures are always given as Percent Accuracy (PA), which is computed from the number of reference labels (N), substitutions (S), insertions (I) and deletions (D) via $PA = 100 \cdot \frac{N-D-S-I}{N}$.

5.1. JAVA system with Token Passing

The *JASPER* system [5] is optimized for noisy speech recognition. At a signal-to-noise-ratio of 0dB in babble noise, its accuracy is 95.7% and 97.4% for the diagonal and the modified imputation (MI) [6] kernel, respectively, when using a 31-dimensional diagonal covariance RASTA-PLP model [13] and 10 dimensional video features. The runtime of its CHMM decoder, averaged over 32 videos of 3s length, is shown on the left side of Tab. 1.

Table 1: Decoding time in ms per 3s of audiovisual data.

mixture components	diag. CPU	MI CPU	diag. GPU	MI GPU
1	95.6	150.6	61.7 (1.5)	65.2 (2.3)
2	179.3	288.3	66.3 (2.7)	63.8 (4.5)
4	298.8	511.5	61.0 (4.9)	67.7 (7.5)
8	565.6	986.8	66.5 (8.5)	70.7 (13.9)
16	1104.4	1945.6	68.4 (16.1)	77.5 (25.1)

In contrast, the right hand columns show the performance with parallelized log-likelihood computations as described by

Section 4.2. In parentheses, the overall speedup factor of the decoding algorithm is given. In addition to this decoding time, a realtime system would need to include feature extraction, which currently requires 2.1s per video in our sequential implementation. Thus, realtime capability is attained only when less than 900ms are spent for CHMM decoding. As can be seen, this is easily achieved for all considered model complexities on the GPU, whereas scalability of sequential computation is limited regarding the complexity of the mixture model and the likelihood kernels.

5.2. WFST decoder using C++

The WFST decoder was used for multi-stream speech recognition in the following experiment. The two combined marginal HMMs were a 39-dimensional full-covariance Mel-frequency Cepstrum model and a 31-dimensional diagonal covariance RASTA-PLP model. The accuracy for the GRID database [10] was 99.3%, both in the JAVA-based reference implementation as well as in the C++-WFST decoder.

Fig. 3 shows the runtime per file, averaged over 47 utterances, for the C++ implementation, where likelihood computation was either performed on the CPU or on the GPU. Similarly

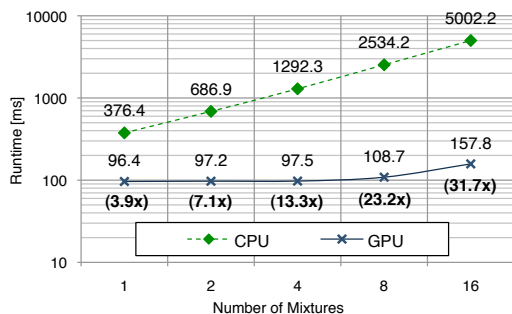


Figure 3: Runtime in ms per file of 3s length, the speedup factor S is given in parentheses.

to the JASPER decoder, the speedup grows almost linearly with model complexity. For the GPU version, system overhead for calling the accelerator dominates the overall runtime for models with less than 4 mixture components. The likelihood computation starts dominating the runtime for more complex models with 8 and 16 mixture components.

6. Conclusion

Coupled HMM decoding is an important technique for robust audiovisual speech recognition as well as for multiple streams of different audio features. While great recognition accuracy improvements are possible, it is computationally expensive and, in general, not suitable for realtime implementations on state-of-the-art CPUs.

To achieve realtime performance and enhance scalability, CHMM decoding can be decomposed into two main algorithmic phases: the likelihood evaluation phase and the search phase. This decomposition enables the design of a system that evaluates the compute intensive likelihood computations on a GPU, while simultaneously carrying out the more memory intensive search operations on the CPU. The decomposition has been applied to two systems: a JAVA audiovisual speech recognizer, and a C++-based WFST decoder that was extended for multistream decoding. Both systems profited greatly from this redesign in terms of speedup, while none of the inherent accuracy was sacrificed.

Our results demonstrate that by using a new generation of highly-parallel commodity processors, CHMM-based audiovi-

sual and multistream speech recognition may be accelerated to achieve real-time human-machine interaction. This is demonstrated even with computationally complex full-covariance or robust likelihood kernels.¹

The use of GPUs can also be extended to the feature extraction operations, which contain rich sources of parallelism in processing the observations both within each time step as well as across time steps. As AVSR achieves faster than real time throughput, it opens up powerful possibilities to include other capabilities such as machine translation and speech synthesis in real-time human-machine interactions. Commercial deployment scenarios of CHMM techniques include more reliable interfaces for bank ATM installations in busy streets, ticket booths at noisy train stations, or more noise-resistant videoconferencing in coffee shops.

7. References

- [1] E. Petajan, B. Bischoff, and D. Bodoff, "An improved automatic lipreading system to enhance speech recognition," in *Proc. SIGCHI Conf. on Human Factors in Computing Systems*, 1988.
- [2] C. Neti, G. Potamianos, J. Luettin, I. Matthews, H. Glotin, D. Vergyri, J. Sison, A. Mashari, and J. Zhou, "Audio-visual speech recognition," Johns Hopkins University, CLSP, Tech. Rep., 2000.
- [3] A. Nefian, L. Liang, X. Pi, X. Liu, and K. Murphy, "Dynamic bayesian networks for audio-visual speech recognition," *EURASIP Journal on Applied Signal Processing*, vol. 11, pp. 1274–1288, 2002.
- [4] G. Papandreou, A. Katsamanis, V. Pitsikalis, and P. Maragos, "Adaptive multimodal fusion by uncertainty compensation with application to audiovisual speech recognition," *Audio, Speech, and Language Processing, IEEE Trans.*, vol. 17, no. 3, pp. 423–435, 2009.
- [5] D. Kolossa, S. Zeiler, A. Vorwerk, and R. Orglmeister, "Audiovisual speech recognition with missing or unreliable data," in *Proc. AVSP*, 2009.
- [6] D. Kolossa, A. Klimas, and R. Orglmeister, "Separation and robust recognition of noisy, convolutive speech mixtures using time-frequency masking and missing data techniques," in *Proc. WAS-PAA*, Oct. 2005, pp. 82–85.
- [7] P. R. Dixon, T. Oonishi, and S. Furui, "Harnessing graphics processors for the fast computation of acoustic likelihoods in speech recognition," *Comput. Speech Lang.*, vol. 23, no. 4, pp. 510–526, 2009.
- [8] K. You, J. Chong, Y. Yi, E. Gonina, C. Hughes, Y.-K. Chen, W. Sung, and K. Keutzer, "Parallel scalability in speech recognition," *Signal Processing Magazine, IEEE*, vol. 26, no. 6, pp. 124–135, 2009.
- [9] P. Cardinal, P. Dumouchel, G. Boulianne, and M. Comeau, "GPU accelerated acoustic likelihood computations," in *Proc. Interspeech*, 2008.
- [10] M. Cooke, J. Barker, S. Cunningham, and X. Shao, "An audiovisual corpus for speech perception and automatic speech recognition," *J. Acoust. Soc. Am.*, vol. 120, no. 5, pp. 2421–2424, 2006.
- [11] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "OpenFst: A general and efficient weighted finite-state transducer library," in *Proceedings CIAA 2007*, ser. Lecture Notes in Computer Science, vol. 4783. Springer, 2007, pp. 11–23.
- [12] S. Young, N. Russell, and J. Thornton, "Token passing: a simple conceptual model for connected speech recognition systems," Cambridge University Engineering Department, Tech. Rep. CUED/FINFENG/TR.38, 1989.
- [13] H. Hermansky and N. Morgan, "RASTA processing of speech," *Speech and Audio Processing, IEEE Trans.*, vol. 2, no. 4, pp. 578–589, 1994.

¹This research is supported in part by an Intel Ph.D. Research Fellowship, by Microsoft (Award #024263), by Intel (Award #024894), and by matching fund from U.C. Discovery (Award #DIG07-10227).