FAST SPOKEN QUERY DETECTION USING LOWER-BOUND DYNAMIC TIME WARPING ON GRAPHICAL PROCESSING UNITS

Yaodong Zhang, Kiarash Adl, James Glass

MIT Computer Science and Artificial Intelligence Laboratory
Cambridge, Massachusetts 02139, USA
{ydzhang, kiarash, glass}@csail.mit.edu

ABSTRACT

In this paper we present a fast unsupervised spoken term detection system based on lower-bound Dynamic Time Warping (DTW) search on Graphical Processing Units (GPUs). The lower-bound estimate and the K nearest neighbor DTW search are carefully designed to fit the GPU parallel computing architecture. In a spoken term detection task on the TIMIT corpus, a 55x speed-up is achieved compared to our previous implementation on a CPU without affecting detection performance. On large, artificially created corpora, measurements show that the total computation time of the entire spoken term detection system grows linearly with corpus size. On average, searching a keyword on a single desktop computer with modern GPUs requires 2.4 seconds/corpus hour.

Index Terms— spoken term detection, GPU, dynamic time warping, CUDA

1. INTRODUCTION

Graphical Processing Units (GPU) have been increasingly used for general purpose computing in the speech research community. As a parallel computing architecture, GPUs are especially designed for dealing with intensive, highly parallel computations that are often encountered in speech tasks. For example, in [1] a large vocabulary speech recognition engine was implemented on GPUs with a significant speed-up observed. [2, 3, 4] have demonstrated how GPUs can be used to rapidly calculate acoustic likelihoods for large mixture models. Sart et al. provided a Dynamic Time Warping (DTW) search algorithm for GPUs in [5], claiming that they could speed up the run time by up to two orders of magnitude.

Inspired by their success, in this paper we propose a fast unsupervised spoken query detection system based on lower-bound K nearest neighbor (KNN) DTW search on GPUs. The lower-bound estimate as well as the KNN-DTW search are carefully designed to utilize GPU's parallel computing architecture. On the TIMIT corpus, a 55x speed-up of a spoken term detection task is achieved when compared to our previous CPU implementation [6], without affecting the detection performance. On a large, artificially created corpus, experiments indicate that the total computation time of the spoken query detection system is linear with corpus size. On average, searching a keyword on a single desktop computer with modern GPUs requires 2.4 seconds/corpus hour.

2. BACKGROUND

2.1. GPU and CUDA

GPUs represent a category of specially designed graphical processors. The main purpose of GPUs is to facilitate fast construction and manipulation of images intended for output to a display. Since images can be decomposed into blocks of pixels and each block can be processed independently, GPUs are designed to contain a highly parallel computing architecture which is more effective than CPUs that are designed for general purpose computing. An additional advantage of modern GPUs connected via a PCI Express interface is that they can access main memory faster than the CPUs themselves [4].

In order to facilitate the fast-growing application of GPUs to non-image processing tasks, NVidia has released the Compute Unified Device Architecture (CUDA) as a library providing APIs for using GPUs to perform general purpose computing [7]. When programming on GPUs with CUDA, CUDA defines a kernel as an abstraction of instances that can be run in parallel. Within each kernel instance, multiple threads can be issued to finish the task assigned to that instance. Therefore, CUDA can be viewed as a two-layer parallel computing architecture. In each CUDA run, one GPU core is assigned to run several kernel instances, and within each kernel instance, multiple threads are running simultaneously [8]. In order to achieve maximum benefit, it is naturally important to take this architecture into consideration when adapting speech processing algorithms to the GPU/CUDA framework.

2.2. Spoken Term Detection using KNN-DTW

We briefly review the unsupervised spoken query detection framework in [6, 9]. Given a spoken query Q with N frames, let $\vec{x}_1, \ldots, \vec{x}_N$ represent MFCCs for each speech

978-1-4673-0046-9/12/$26.00 ©2012 IEEE

frame. A D-mixture GMM G is trained on all N frames without using any labels. Subsequently, for each speech frame, $\vec{x}_i$, a posterior probability, $p_i^j = P(g_j|\vec{x}_i)$, can be calculated where g_j denotes j-th Gaussian component in GMM G. Collecting D posterior probabilities, each speech frame $\vec{x}_i$ is then represented by a posterior probability vector $\vec{p}_i = \{p_i^1, \ldots, p_i^D\}$ called a Gaussian posteriorgram. After representing the spoken query and speech documents using Gaussian posteriorgrams, an efficient KNN-DTW is used to find the top K nearest neighbor document matches. Specifically, if $Q = \{\vec{q}_1, \ldots, \vec{q}_M\}$ denotes the query posteriorgram and $S = \{\vec{s}_1, \ldots, \vec{s}_N\}$ denotes a speech segment, an upper-bound envelope sequence U is calculated on Q, where $U = \{\vec{u}_1, \cdots, \vec{u}_M\}$, $\vec{u}_i = \{u_i^1, \cdots, u_i^D\}$ and $u_i^p = \max(q_{i-r}^p, \cdots, q_{i+r}^p)$. U can be viewed as a sliding-maximum on Q with window size r. Then, a lower-bound estimate of the DTW distance DTW(Q, S) is computed by

$$L(Q,S) = \sum_{i=1}^{l} d(\vec{u}_i, \vec{s}_i) \tag{1}$$

where $l = \min(M, N)$ and $d(\cdot)$ represents an inner product distance. If there are C speech segments, C lower-bound estimates are calculated and ranked. The KNN search starts from the segment with the smallest lower-bound estimate and performs DTW alignment. In [6] we proved that $L(Q, S) \leq DTW(Q, S)$, guaranteeing that the KNN search can stop when the current lower-bound estimate is greater than the DTW distortion score of the K^{th} best match.

3. SYSTEM DESIGN

In this section, we describe the proposed parallel spoken query detection system. The entire implementation consists of four CUDA kernels. The first two kernels correspond to implementation of a parallel lower-bound estimate, while the last two kernels are used for parallel DTW calculations. Since we believe that the proposed method can be implemented in parallel architectures [10] other than CUDA, our focus will be on describing how to decompose the framework into parallel modules rather than using too many CUDA specific terms (e.g. grid, block). Moreover, since CUDA can run multiple kernel instances on one GPU core, in order to avoid confusion and without loss of generality, in the following description, we assume that each kernel instance runs on one GPU core. Figure 1 illustrates the parallel implementation.

3.1. Parallel Lower-bound Estimate

The parallel lower-bound estimate computation consists of two kernels. The first kernel computes the frame-wise inner-product distance, while the second kernel sums the inner-product distances. Specifically, suppose the upper bound envelope of the spoken query Q is U with length l and one of the speech segments is S_j with length l. By applying Eq. (1), the lower-bound estimate of Q and S_j is essentially the summation of the frame-by-frame inner-product distance of U and S_j. The summation calculation can be divided into two independent steps. In the first step, the inner-product distance d_i for each frame pair $\vec{u}_i$ and $\vec{s}_i$ is computed. The second step sums over l distances to obtain the lower-bound estimate.

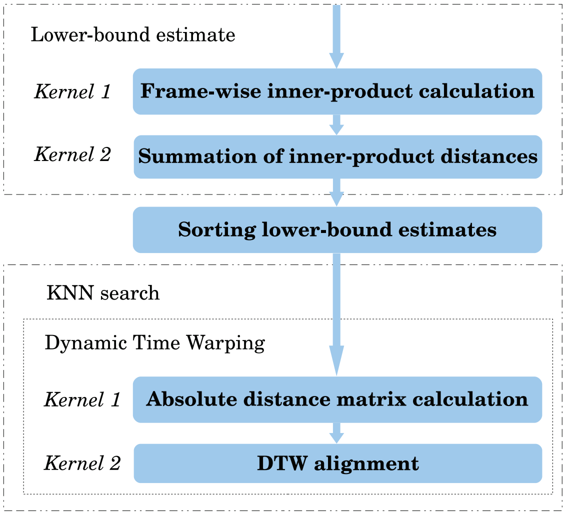
Lower-bound estimate
Kernel 1
Frame-wise inner-product calculation
Kernel 2
Summation of inner-product distances
Sorting lower-bound estimates
KNN search
Dynamic Time Warping
Kernel 1
Absolute distance matrix calculation
Kernel 2
DTW alignment

Fig. 1. System flowchart

In order to maximize GPU efficiency, two separate kernels are designed for each step respectively. The first kernel takes two posteriorgram vectors as input, and outputs the inner-product distance of these two vectors. Since each kernel can run multiple threads simultaneously, the vector-wise inner-product calculation is further decomposed into element-wise products. Since each posteriorgram is a D-dimensional vector, the inner-product can be written as

$$d(\vec{u}_i, \vec{s}_i) = \sum_{k=1}^{D} u_i^k \cdot s_i^k. \tag{2}$$

Therefore, as illustrated in Figure 2, each thread in the kernel corresponds to one product, so that the D products can be computed simultaneously. After the D threads finish computing, the summation of D products is parallelized by using a parallel-sum-reduction method [8]. When the first kernel finishes computing, l inner-product distances are stored in on-device memory. The second kernel then launches to sum over l inner-products. This summation is also performed via the parallel-sum-reduction technique.

3.2. Sorting and KNN Search

After the lower-bound estimates have been computed, each speech segment S_j is associated with a lower-bound distance estimate LB_j. Since the number of the target speech segments can potentially be very large, a CUDA library called

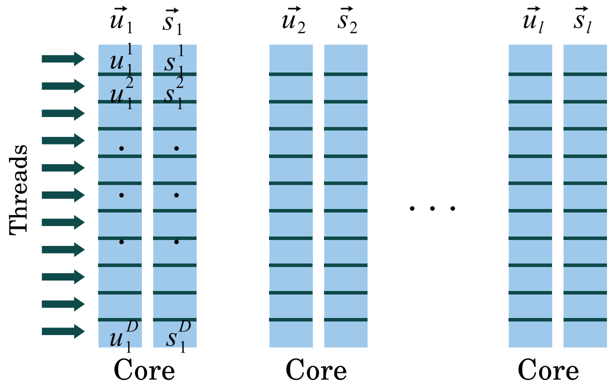

Fig. 2. Parallel frame-wise inner-product calculation

Thrust is used to perform high-performance parallel sorting of these lower-bound estimates [11].

When sorting is complete, the KNN search starts from the speech segment with the smallest lower-bound estimate and calculates the actual DTW distance. In theory, in order to obtain K best matches, at least K DTW alignments need to be performed [6]. Therefore, instead of going through the sorted speech segments one by one in the CPU implementation and calculating one DTW alignment at a time, K speech segments are considered and K DTW alignments are performed simultaneously in the GPU implementation. If the current best DTW matching score is less than the lower-bound value of the $(K+1)^{th}$ speech segment, the KNN search terminates and outputs the best K matches. Otherwise, the next $K/2$ speech segments are considered and $K/2$ new DTW alignments are computed. The search iterates until the termination condition is met. In summary, the parallel KNN search strategy is to aggressively run K DTW alignments in the first round of search, and subsequently to perform $K/2$ DTW alignments to be more conservative and avoid wasting DTW calculations.

3.3. Parallel DTW

In order to maximize the use of the parallel computing architecture, DTW calculations are divided into two steps. In the first step, given two posteriorgrams Q and S_j, an absolute distance matrix $A(Q, S_j)$ (which uses the inner-product as local distance metric) is computed. The second step performs the DTW alignment on A and outputs the distortion score. A kernel is designed for each respective step.

Specifically, if Q and S_j have the same length l, the size of the absolute distance matrix $A(Q, S_j)$ is $l \times l$. Each instance of the first kernel computes a single element in A. Note that because of the DTW beam width constraint, only valid elements are computed. Since the input to the first kernel is two D-dimensional posteriorgram vectors, in a manner similar to the first kernel in the previous step, each thread in each kernel instance computes a scalar product so that D products can be performed simultaneously. Since the DTW alignment is a 2-D sequential dynamic search, it cannot be completely parallelized. Therefore, in the second kernel, each thread in each kernel instance corresponds to the entire warping process given an absolute distance matrix A.

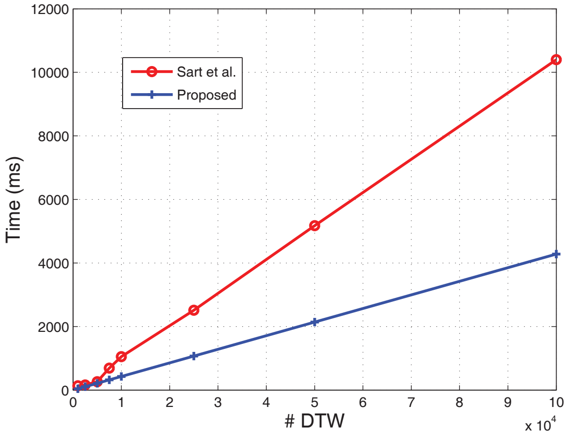

Fig. 3. Comparison of computation time for parallel DTW

In [5], the DTW GPU implementation was to use each thread in each kernel instance to compute the absolute distance matrix A as well as the DTW alignment. In order to compare the two approaches, we implemented both methods and report the results of the comparison in the next section.

4. EVALUATION

4.1. Spoken Term Detection Task

The spoken query detection task was evaluated on the 630 speaker TIMIT corpus that includes a training set of 3,696 utterances and a test set of 944 utterances. As in [6], 10 query keywords were randomly selected and extracted from the training set. For each keyword example, the query detection task was to rank all 944 utterances from the test set based on the utterance's possibility of containing that keyword. Overall performance was measured by the average equal error rate (EER): the average rate at which the false acceptance rate is equal to the false rejection rate.

All spoken term detection experiments were performed on a computer equipped with an Intel® Quad 6600 Processor (2.66 GHz, 4 cores), 8 GB RAM and a NVidia® GTX 580 graphic card (512 GPU cores with 1.5 GB on-board RAM). The graphic card cost 500 USD in June, 2011.

4.2. Experimental Results

In order to validate the correctness of the GPU implementation of the spoken query detection algorithms, we first compared the detection results with the corresponding CPU implementation. In addition to the intermediate outputs from the four kernels being the same, the overall detection EER remained the same as has been previously reported [6].

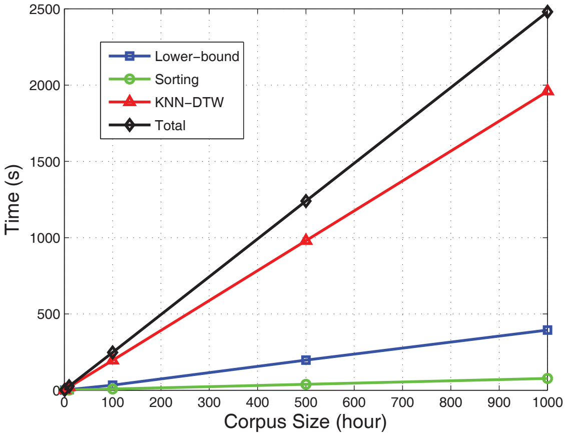

Fig. 4. Decomposition of computation time vs. corpus size

Figure 3 shows a comparison of the computation time for the two different aforementioned parallel DTW implementations. In the figure, the x-axis shows the number of DTWs computed, the red curve represents the time consumed for the method proposed by Sart et al. [5], while the blue curve shows the method used in the current implementation. The figure indicates that both methods consumed similar time when computing less than 5,000 DTWs. However, our method outperformed the previous method by a factor of 2 in terms of the running time when more than 5,000 DTWs computations were required. We believe the reason for the greater efficiency is due to the time saved in the absolute distance matrix calculation. In the current version the entire matrix was decomposed into individual elements for maximum parallelization, while the prior method computed the full matrix in each thread without parallelization. As the number of DTW calculation increases, the amount of computation time saved becomes more and more apparent.

Figure 4 shows the average computation time needed for searching one keyword using the proposed spoken query detection system as a function of corpus size. Since the original TIMIT test corpus only contains 48 minutes of speech, in order to measure the computation time on a larger corpus, we replicated the TIMIT test corpus by adding small amounts of random noise into each speech frame. The replicated TIMIT test corpus contained 1,000 hours of speech. In the figure, the black curve represents the total time consumed, the blue curve represents the time consumed by the lower-bound estimate, the green curve represents the time used by sorting all lower-bound values, and the red curve represents the time consumed by the KNN-DTW search. The results indicate that the KNN-DTW search occupies nearly 80% of the total running time, which we believe is due to the difficulty in parallelizing the DTW algorithm. It is encouraging to observe that the total computation time grows linearly with the corpus size. For 1,000 hours of speech, searching a keyword requires 40 minutes on a single desktop computer, which translates to 2.4 seconds/corpus hour. Note that with multiple desktops and GPUs, the entire process could be further parallelized with each GPU searching a subset of the overall corpus.

In terms of computation time for searching the TIMIT test corpus, the original CPU-based approach [6] took, on average, 120 seconds per keyword, while the GPU-based approach takes, on average, only 2.2 seconds per keyword, which translates to a speed-up factor of 55.

5. CONCLUSION AND FUTURE WORK

In this paper we have described a parallelized implementation of an unsupervised spoken query detection system based on lower-bound KNN-DTW search, and tested the implementation on Graphical Processing Units (GPUs). The spoken query detection algorithm is carefully re-designed to fit GPU's parallel computing architecture. In a spoken query detection task using the TIMIT corpus, a 55x speed-up is achieved compared to our previous CPU-based implementation without affecting the detection performance. On artificially replicated data, experimental results indicate that the total running time of the entire spoken query detection system grows linearly with corpus size. On average, searching a keyword on a single desktop computer with modern GPUs requires 2.4 seconds/corpus hour.

In the future, we plan to investigate additional GPU efficiencies, implementations using GPU farms, as well as other parallel computing architectures such as FPGAs [5].

6. REFERENCES

[1] J. Chong, E. Gonina, Y. Yi, and K. Keutzer, "A fully data parallel WFST-based large vocabulary continuous speech recognition on a graphics processing unit.," in *Proc. INTERSPEECH*, 2009, pp. 1183–1186.

[2] P. Cardinal, P. Dumouchel, G. Boulianne, and M. Comeau, "GPU accelerated acoustic likelihood computations.," in *Proc. INTERSPEECH*, 2008, pp. 964–967.

[3] P. R. Dixon, T. Oonishi, and S. Furui, "Fast acoustic computations using graphics processors," in *Proc. ICASSP*, 2009, pp. 4321–4324.

[4] J. Vanek, J. Trmal, J. V. Psutka, and J. Psutka, "Optimization of the gaussian mixture model evaluation on GPU," in *Proc. Interspeech*, 2011, pp. 1737–1740.

[5] D. Sart, A. Mueen, W. Najjar, E. Keogh, and V. Niennattrakul, "Accelerating dynamic time warping subsequence search with GPUs and FPGAs," in *Proc. ICDM*, 2010, pp. 1001–1006.

[6] Y. Zhang and J. Glass, "An inner-product lower-bound estimate for dynamic time warping," in *Proc. ICASSP*, 2011, pp. 5660–5663.

[7] "NVidia CUDA Toolkit 4.0," `http://developer.nvidia.com/cuda-toolkit-40/`.

[8] "CUDA C Best Practices Guide," `http://developer.nvidia.com/cuda-downloads/`.

[9] Y. Zhang and J. Glass, "Unsupervised spoken keyword spotting via segmental DTW on Gaussian posteriorgrams," in *Proc. ASRU*, 2009, pp. 398–403.

[10] "ATI Stream," `http://www.amd.com/stream/`.

[11] J. Hoberock and N. Bell, "Thrust: A Parallel Template Library," `http://www.meganewtons.com/`.