Online Unsupervised Pattern Discovery in Speech using Parallelization

Mrugesh R. Gajjar†, R. Govindarajan†, T. V. Sreenivas‡

†Supercomputer Education & Research Centre
‡Department of Electrical Communication Engineering
Indian Institute of Science, Bangalore, India - 560012

gajjar@hpc.serc.iisc.ernet.in, govind@serc.iisc.ernet.in, tvsree@ece.iisc.ernet.in

Abstract

Segmental dynamic time warping (DTW) has been demonstrated to be a useful technique for finding acoustic similarity scores between segments of two speech utterances. Due to its high computational requirements, it had to be computed in an offline manner, limiting the applications of the technique. In this paper, we present results of parallelization of this task by distributing the workload in either a static or dynamic way on an 8-processor cluster and discuss the trade-offs among different distribution schemes. We show that online unsupervised pattern discovery using segmental DTW is plausible with as low as 8 processors. This brings the task within reach of today's general purpose multi-core servers. We also show results on a 32-processor system, and discuss factors affecting scalability of our methods.

Index Terms: Unsupervised pattern discovery, Dynamic time warping, Parallelization, Spoken language systems

1. Introduction

With the explosion of digital multimedia content, either on the Internet or institutional/personal collections, new problems related to speech and audio processing have emerged. Examples include, audio information retrieval [1], audio and speech mining, keyword spotting [1], automated surveillance and monitoring [2]. Various machine learning techniques are applied on the multimedia content for different applications, of which automatic speech recognition (ASR) is a common application. The commonly addressed ASR is in a supervised manner, i.e., for a known language, vocabulary, speaker, acoustic environment, etc. There are unsupervised applications also, where parameters are unknown. The unsupervised machine learning problems occur in other domains as well, such as genomics.

Recently, Park et al. [3, 4] have reported keyword discovery from academic lectures in an unsupervised manner. Using segmental dynamic time warping (DTW) technique they search for similar acoustic patterns in the lecture and then use clustering to extract matching segments, finally running a phone recognizer to find the transcription. This is a data mining type of problem in speech signals and has been addressed for different applications [3][5]. Segmental DTW approach is computationally very intensive and for analyzing large speech databases such as lectures, conferences, the processing is proposed to be done in an *off-line* manner [3]. We believe that unsupervised pattern discovery techniques can be very useful if they are made to work in an online manner, introducing some component of spontaneity. For example, this can be useful in surveillance devices deployed in broadcast monitoring for security, and other non-speech applications also. Unsupervised pattern discovery can also be useful in security applications where the speaker language may be unknown. In the commercial domain, we require automatic news broadcast monitoring and summarization services [6]. Recently, Xie [7] performed unsupervised pattern discovery experiments on broadcast multimedia content such as sports and news and detected high-level temporal events such as play or breaks. Clearly one requires huge computational resources to mine multimedia content in greater detail. Network security/streaming also requires certain online pattern discovery, which would involve huge computation and would be benefited by parallelization.

In this paper, we explore real-time or near real-time implementation of the segmental DTW algorithm. Like many signal processing algorithms, the segmental DTW algorithm is also developed in a sequential processing framework, executed on a single processor. Since the algorithm is computationally intensive, it is important to explore a parallelized version for on-line applications. The current trend in semiconductor technology enables multiple processors or cores on a chip [8]. Technology predictions indicate few tens of cores in a single chip in the near future. With such multi-core architectures, parallelization of sequential algorithms would be more effective. In the past, parallelization has been used either to improve the accuracy of ASR [9], or to speed up recognition time [10]. Parallelization, in general, is a well addressed topic and several standard schemes exist [11]. We explore two parallelization schemes viz., static and dynamic. However, while using these techniques, consideration of the characteristics of the program being parallelized becomes an important factor for achieving better performance. We incorporate segmental DTW specific knowledge in the parallelization schemes. We show that for this task, it is plausible to achieve online performance with as low as 8 processors. With 8-core general purpose server system on a single board, we need not go beyond one compute node for such kind of unsupervised pattern discovery in an online manner. We propose and evaluate both static and dynamic parallelization (workload distribution) schemes on 4, 8 and 32 processors for a sample workload of 40 minutes long lecture speech data. We get best speed-ups of 3.66, 7.38 and 21.73 respectively. One of our static parallelization schemes (*bcast-eq-frames*) works nearly best across all configurations, while dynamic schemes reach bandwidth bottleneck on 32 processors.

2. Sequential Segmental DTW

Dynamic Time Warping (DTW) is a technique for optimally aligning variable length sequences of symbols based on some distance measure between the symbols [12]. Segmental DTW [3] is a technique for finding sub-sequences having low-cost alignments from given sequences of symbols. In our context, it

PROCEDURE sequential_segmental_DTW
Given a set of Speech segments S and parameter W,
for each $(i, j) \in (S \times S), i < j,$

- *call* segmental_DTW(i,j,W)

endfor

Figure 1: Sequential segmental DTW

takes two speech utterances (segments[1]) and finds all low-cost alignment paths within the utterances. From now on we use the term *sequential segmental DTW* to refer to the sequential algorithm for calculating segmental DTW between all speech segment pairs.

Figure 1 shows the sequential segmental DTW algorithm which calculates segmental DTW for each pair of speech segments. Note that the procedure *segmental_DTW* is symmetric with respect to segments, that is, scores of *segmental_DTW*(i, j) would be equal to that of *segmental_DTW*(j, i) for all i, j. So we only need to calculate *segmental_DTW* for unique unordered pairs of segments. For more details on *segmental_DTW* procedure refer to [3].

Our approach to *parallel segmental DTW* essentially distributes the workload of all speech segment pairs among multiple processors that calculate segmental DTW on their assigned pairs. Note that our approach does not parallelize the procedure *segmental_DTW(i,j,W)* where *i,j* are speech segments and *W* is the width parameter to constrain the DTW search.

3. Parallelization Schemes

In this section we present an overview of the parallelization schemes. For any parallelization scheme to be effective all the processors should complete their work at the same time. Any scheme lacking in the requirement above will result in load imbalance as a result of some processors being idle. The schemes can be classified as either static or dynamic based on the distribution of workload. In a dynamic scheme, the workload to a processor is decided at runtime while in a static scheme, at the startup one processor assigns workload to each processor and then all processors work on the assigned tasks. In this paper, we evaluate two versions of each type of parallelization scheme specific to the problem of distributing the workload of *sequential segmental DTW*.

In our dynamic scheme, master-slave strategy, one processor acts as master processor which distributes a small chunk of work to other slave processors. A Slave processor returns to master for more work once it has completed the assigned chunk. Master processor releases slave processors when there is no more work left.

In the static approach, the whole workload is initially divided equally according to some measure and assigned to each processor with its part of workload. All the processors start processing their respective workload and there is no need for communication among the processors except until at the end when they synchronize and exit. Our two versions of this strategy differ in the measure we choose for dividing the workload equally. In the following subsections we describe specifics of our implementation.

[1] different in meaning from segment in *segmental* DTW

PROCEDURE parallel_segmental_DTW_master
Given a set of Speech segments S and parameter W,
for each $(i, j) \in (S \times S), i < j,$

- wait for $ready$ slave process
- send (i, j, W) to $ready$ slave process

endfor

Figure 2: Master process of Master-slave scheme

PROCEDURE parallel_segmental_DTW_slave
while(true)

- announce readiness to $master$ process
- receive (i, j, W) from $master$ process
- *call* segmental_DTW (i, j, W)

endwhile

Figure 3: Slave process of Master-slave scheme

3.1. Master-slave strategy (Dynamic schemes)

In our first scheme, the chunk of work the master assigns every time a slave process becomes ready is a pair of speech segments. Slave calculates $segmental\_DTW$ on assigned speech segments and announces the completion to master. As $segmental\_DTW$ procedure is symmetric, the master process basically iterates over all unordered pairs of speech segments and keeps on assigning them to slaves until it finishes all pairs. The master process does not perform any useful work except sending segment pairs to slaves. The master and slave algorithms are listed in Figure 2 and Figure 3 respectively.

We note here that load imbalance would be minimized if the master processor, sorts (from biggest to smallest) all speech segment pairs according to their length in a queue and then start distributing from the head of the queue. This way larger tasks get assigned initially and at the end smaller tasks tend to make all slave processors complete at the same time. This is our second dynamic scheme.

3.2. Broadcast based uniform data distribution (Static schemes)

In this approach we want to statically divide the workload equally among the processors and send it to them. Here our workload consists of unordered pairs of speech segments. Our workload can be visualized as a symmetric matrix where only half of the values need to be calculated. For p processors, one solution is to divide the matrix into p vertical (or equivalently horizontal) strips, and let each processor only compute the shaded portions of the strip as shown in Figure 5a. As seen from the strips, as shaded regions in any of the strip always span all the segments, one process broadcasts all the segments initially along with the width of the strip (number of segments per process which is same for all the processes) to all processes. However, practically this load distribution results in load imbalance in processors. Execution time of $segmental\_DTW$ procedure is dependent on the length in number of frames of speech segments and in practice, segments have non-uniform distribution of frames among them. The load imbalance delays the parallel program completion.

In our second approach, we size each strip's width such that the total length of segments (in number of frames) in each strip is roughly equal. The resulting load distribution scheme can be

PROCEDURE broadcast_segmental_DTW
Given a set of Speech segments S, set of processors P, parameter W and current process' id p,
if(p==1) broadcast S
if(p==1)
for each i in P

- $S_i^{'} = determine\_chunk(S, P, i)$
- broadcast ($\mid S_i^{'} \mid, i$)
- send indices of $S_i^{'}$ chunk of segments to process i

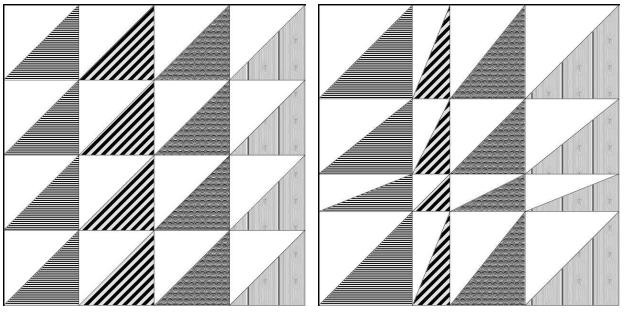
endfor
Receive this process' chunk $S_p^{'}$
for each $(i, j) \in (S^{'} \times S)$

- b=$\sum_{m=0}^{x} \mid S_m^{'} \mid, x = argmax \left[\left\{ \sum_{l=0}^{k} \mid S_l^{'} \mid \right\} < j \right]$
- $j^{'} = j\%$b
- ***if***$((i\% \mid S_p^{'} \mid) > j^{'})$
  call segmental_DTW(i, j, W)

endfor

Figure 4: Broadcast based uniform data distribution scheme

seen in Figure 5b, where the strips need not be of equal width (in number of segments). However, with the non-equal strip width exploiting the symmetry in the matrix and avoiding redundant computation is not straight forward. Figure 5b illustrates how our approach exploits the symmetry. We need to broadcast strip widths to be able to calculate shaded portions shown in Figure 5b. The detailed algorithm is given in Figure 4.



(a) Equal number of Segments per processor (b) Equal number of Frames per processor

Figure 5: Illustration of four processors calculating segmental DTW for speech segment pairs. Each processor handles a column and need to calculate only the segment pairs whose coordinates fall in the shaded portion.

4. Experimental Results

4.1. Experimental Setup

We ran the parallel programs on two different clusters.

1. An 8 Processor (Itanium 2 1.6 GHz) Linux cluster having four 2-way-SMP nodes with 4 GB RAM/node with Myrinet and Gigabit Ethernet as interconnects.

2. A 32 Processor (IBM Power5 1.6 GHz) Linux cluster with eight 4-way-SMP nodes with 4 GB RAM/node and Gigabit Ethernet.

Processors communicate by passing messages using MPI (Message Passing Interface) library. All programs were compiled using the highest compiler optimization level.

Sequential, no. of processors=1, Execution time (et)=65.2 min						
Scheme	no. of proc.=4			no. of proc.=8		
	et	s	η	et	s	η
ms	23	2.83	0.71	10.5	6.21	0.78
ms-sorted	21.3	3.06	0.77	9.7	6.72	0.84
bcast-eq-seg	18	3.62	0.91	11.1	5.87	0.73
bcast-eq-frames	18	3.62	0.91	9.3	7.01	0.88

Table 1: 40 min lecture, Frame size 10 ms, Features/frame=13

Sequential, no. of processors=1, Execution time (et)=89.9 min						
Scheme	no. of proc.=4			no. of proc.=8		
	et	s	η	et	s	η
ms	32.7	2.75	0.69	15.5	5.8	0.73
ms-sorted	28	3.21	0.80	13.5	6.66	0.83
bcast-eq-seg	25.1	3.58	0.90	15.2	5.91	0.74
bcast-eq-frames	24.7	3.64	0.91	12.8	7.02	0.88

Table 2: 40 min lecture, Frame size 10 ms, Features/frame=39

Speech input data: We use a lecture on physics which is 40 minutes long 16K samples/sec, 16 bit/sample wav file. We converted it to 13-dimensional and 39-dimensional MFCC (Mel frequency cepstral co-efficients) feature vectors by the Hidden Markov Model Toolkit (HTK) [13]. We used 25 ms hamming window with 5 ms spacing. We also report results for 10 ms spacing. Single precision floating point values were used for feature vector elements. Then we remove silent regions by using thresholding based on signal power (30 dB). After thresholding we get 1158 *speech segments* totaling 421322 frames among them for 5 ms frame size. For 10 ms frame size, we get 1113 *speech segments* totaling 210684 frames.

4.2. Experimental Results on 8 processor cluster

We report execution time for using different number of processors for the following schemes:
(i) master-slave parallelization scheme, (*ms*)
(ii) master-slave with sorted speech segment queue (*ms-sorted*)
(iii) broadcast scheme with segments distributed uniformly, (*bcast-eq-seg*) and
(iv) broadcast scheme with frames distributed uniformly, (*bcast-eq-frames*)
We report the execution time et (in minutes), the speedup s, defined as the ratio of execution time on a one processor to that on a 'n' processor cluster, and efficiency η, defined as the ratio of speedup to the number of processors. Performance results of our experiments on 8 processor cluster are summarized in Tables 1- 4. We can see from these tables that *bcast-eq-frames* scheme outperforms the other schemes, as it balances the load better. The *ms* scheme performs poorly as it does not utilize the master processor in computations and this affects performance when the number of processors is low. The *ms-sorted* scheme performs always better than *ms* scheme for both 4 and 8 processors.

Sequential, no. of processors=1, Execution time (et)=321.4 min						
Scheme	no. of proc.=4			no. of proc.=8		
	et	s	η	et	s	η
ms	110.3	2.91	0.73	47.7	6.74	0.84
ms-sorted	102.5	3.14	0.78	45.1	7.13	0.89
bcast-eq-seg	96.2	3.34	0.84	52	6.18	0.77
bcast-eq-frames	88.4	3.64	0.91	46	6.99	0.87

Table 3: 40 min lecture, Frame size 5 ms, Features/frame=13

Sequential, no. of processors=1, Execution time (et)=415.7 min						
Scheme	no. of proc.=4			no. of proc.=8		
	et	s	η	et	s	η
ms	143.3	2.9	0.73	63.3	6.57	0.82
ms-sorted	125.9	3.3	0.83	56.3	7.38	0.92
bcast-eq-seg	113.9	3.65	0.91	66.6	6.24	0.78
bcast-eq-frames	113.7	3.66	0.92	58.9	7.06	0.88

Table 4: 40 min lecture, Frame size 5 ms, Features/frame=39

4.3. Experimental Results on 32 processor cluster

The performance of different parallelization schemes for 32 processor system is reported in Table 5. It can be seen that the static schemes perform significantly better than the dynamic master-slave schemes. This is because, as the number of processors increases, load balancing needs to be more fine grained and more tuned to the application characteristics. We can observe dynamic strategies like master-slave are most inefficient for 32 processors. Further, the bandwidth at the master processor also becomes the bottleneck and the master processor is not able to serve slaves as soon as the slaves complete their work. This introduces waiting time in slave processors that causes loss in efficiency. The efficiencies are lower for 39 dimension feature vectors than 13 dimension feature vectors as the former case requires transmission of 3 times the data required for the latter case. Naturally bandwidth at master will become the bottleneck. To alleviate this problem, in higher number processors we must go toward using possible multiple master processors and making the algorithm completely distributed, one special case of that is our static distribution of data.

4.4. Discussion

The execution times reported in Table 1 and 2 show that calculation of segmental DTW for 10 ms frames achieves real-time performance. We note that even on 4 processors it is real-time. From Table 3, 5 ms frames task is near real-time for 13 dimensional feature vector per frame on 8 processors. Thus it is possible to run these algorithms on an 8-core server to achieve online processing. However, making these algorithms exactly online requires overlapping of streaming speech and computation of segmental DTW on extracted segments. The main challenge is that the number of unordered pairs of segments increases quadratically with time. In future, we wish to work on that problem. We would also like to look at parallelizing for multiple channels on higher number of processors and implementation of our schemes on multi-core processors.

No. of processors=32						
Scheme	Features/frame=13			Features/frame=39		
	Seq. et=347.7 min			Seq. et=384.9 min		
	et	s	η	et	s	η
ms	21.3	16.32	0.51	37.5	10.26	0.32
ms-sorted	23.3	14.92	0.47	37.2	10.35	0.32
bcast-eq-seg	16	21.73	0.68	18.8	20.5	0.64
bcast-eq-frames	16	21.73	0.68	18.5	20.8	0.65

Table 5: 40 min lecture, Frame size 5 ms

5. Summary

We parallelized the computationally intensive segmental DTW algorithm for efficient processing of large amount of speech data. We have demonstrated two variations of static and dynamic parallelization schemes on two different clusters. We propose efficient technique for static distribution of work so as to reduce load imbalance on processors without any redundant computations. Experimental results indicate that proposed schemes work well for irregular computations like segmental DTW. Further, our results indicate that the computational requirement of online unsupervised pattern discovery is within the reach of multi-core servers of today.

6. References

[1] Jonathan Foote, "An overview of audio information retrieval," *Multimedia Syst.*, vol. 7, no. 1, pp. 2–10, 1999.

[2] G. Valenzise, L. Gerosa, M. Tagliasacchi, F. Antonacci, and A. Sarti, "Scream and gunshot detection and localization for audio-surveillance systems," in *IEEE Conference on Advanced Video and Signal Based Surveillance, AVSS 2007*, pp. 21–26.

[3] A. Park and J. R. Glass, "Towards unsupervised pattern discovery in speech," in *Proc. IEEE ASRU 2005*, Nov./Dec. 2005, pp. 53–58.

[4] A. Park and J. R. Glass, "Unsupervised word acquisition from speech using pattern discovery," in *Proc. IEEE ICASSP 2006*, May 2006.

[5] A. Park and J. R. Glass, "A novel dtw-based distance measure for speaker segmentation," in *Proc. SLT 2006*, Dec. 2006.

[6] J. Makhoul, F. Kubala, T. Leek, Daben Liu, Long Nguyen, R. Schwartz, and A. Srivastava, "Speech and language technologies for audio indexing and retrieval," *Proceedings of the IEEE*, vol. 88, no. 8, pp. 1338–1353, Aug 2000.

[7] L. Xie., "Unsupervised pattern discovery for multimedia sequences," *Ph.D Thesis, Columbia University.*, 2005.

[8] Lance Hammond, Basem A. Nayfeh, and Kunle Olukotun, "A single-chip multiprocessor," *Computer*, vol. 30, no. 9, pp. 79–85, 1997.

[9] Salvatore J. Stolfo, Zvi Galil, Kathleen McKeown, and Russell Mills, "Speech recognition in parallel," in *Proc. of HLT '89*, 1989, pp. 353–373.

[10] M. Ravishankar, "Parallel implementation of fast beam search for speaker-independent continuous speech recognition," *Technical Report, Computer Science and Automation, Indian Institute of Science, Bangalore, India.*, 1993.

[11] Ananth Grama, Vipin Kumar, Anshul Gupta, and George Karypis, *Introduction to Parallel Computing*, Addison-Wesley, 2003.

[12] Lawrence Rabiner and Biing-Hwang Juang, *Fundamentals of speech recognition*, Prentice-Hall, Inc., 1993.

[13] "The hidden markov model toolkit (HTK)," *http://htk.eng.cam.ac.uk/*.