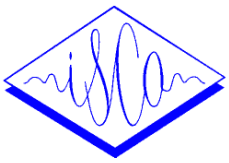

3rd European Conference on
Speech Communication and Technology
EUROSPEECH'93
Berlin, Germany, September 19-23, 1993

PARALLELISING K-MEANS CLUSTERING ON DISTRIBUTED MEMORY MIMD COMPUTERS

J.A. Elliott, M.E. Forsyth, F.R. McInnes and N.W. Ramsey

Centre for Speech Technology Research, University of Edinburgh, 80 South Bridge, Edinburgh, EH1 1HN, Scotland, UK

ABSTRACT

Two strategies are presented for the parallelisation of the k-means clustering algorithm on MIMD distributed-memory computers (multicomputers). The implementations are decribed in detail. The advantages of each outlined.

Keywords: *Speech recognition, VQ Codebook, k-means clustering, parallel processing*

1 INTRODUCTION

This paper presents two strategies for implementing the k-means clustering algorithm on parallel computers. The algorith and system are outlined in Section 2. Section 3 gives a computational analysis of the k-means algorithm. Section 4 describes the methods used to achieve the mapping onto parallel processors. Section 6 concludes the paper.

2 OVERVIEW OF THE SYSTEM

The aim of a clustering algorithm is to produce a set of clusters which best represent a set of training vectors. These clusters can in turn be represented by their mean vectors (or centroids). The centroids can then be used to form a VQ codebook.

The k-means algorithm (or variations thereupon [1, 2]) has been used for several years as a means of unsupervised clustering of data (Fig. 2). It has been especially popular in the field of speech recognition as a means of producing vector quantisation (VQ) codebooks (Fig. 1). Another potential application in the same field is that of speaker adaption: speech data may slowly change as the talker's speech habits change. An on-line clustering algorithm may be used to update the pattern library so that it can track the slowly varying changes. Clearly in this case the models used by the recogniser would also need to be retrained.

The k-means clustering algorithm is highly computationally intensive; it can take many hours, even a few days, to cluster a large set of training vectors using today's uniprocessor workstations. It is beneficial to have a fast means of clustering so that more experiments in VQ codebook design can be carried out in the time available. In the case of speaker adaption it would be possible to build a system that reacted to the speaker in real time. To this end this paper proposes two strategies for parallelising the k-means algorithm on today's low-cost readily-available multicomputers, e.g. a few transputers added on to a PC or workstation.

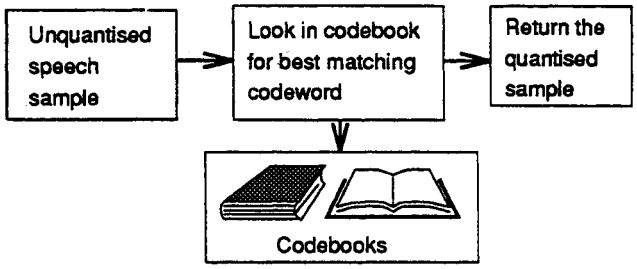

Fig. 1: Vector quantisation

3 ANALYSIS OF THE ALGORITHM

For this analysis it will be assumed that there are N training vectors and K classes for clustering. The k-means algorithm (Fig. 2) consists of three main parts: a) Classification of training vectors; b) Computing cluster centroids; c) Testing for convergence. It is the first two parts which require the vast majority of processing power. They both typically involve some kind of distance measure to be calculated between all the vectors. For speech recognition these may be cepstral vectors in 12-D space. Thus a Euclidean distance measure involves at least 12 floating-point subtractions and multiplications.

The k-means algorithm has computational complexity $O(NK)$. Profiling a serial implementation of the algorithm determined that the vast majority of computation is used re-classifying the training vectors once the new cluster centroids have been determined. Any successful parallelisation of the algorithm must distribute this classification effort as evenly as possible amongst the available processors.

1. **Initialise:** *Put all the training vectors in one cluster and find its centroid vector. The number of clusters is one i.e.* $L = 1$.
2. **Split:** *All clusters are divided in two, making the size of the cluster set,* $L = 2L$. *The centroids for the new clusters are found.*
   (a) **Classification:** *The training vectors each are re-assigned to the 'nearest' new cluster centroid.*
      - **Find centroids:** *Find the new centroids for the cluster set, based upon the new training vector distribution.*
   (b) **Test convergence:** *If the cluster centroids have all converged, goto Step 3. Otherwise, goto Step (a).*
3. **Test finished:** *If* L *is equal to the desired codebook size,* K, *STOP. Otherwise, goto Step 2.*

Fig. 2: Traditional k-means algorithm.

4 MAPPING PROCESSES TO PARALLEL PROCESSORS

It is traditional to parallelise an algorithm by first implementing a serial version which identifies the necessary processes. Next, it must be decided how to distribute these serial processes across the available processors. This task is known as the 'mapping problem' [4]. The goodness of the mapping determines directly the speedups achieved. It is quite possible to choose a mapping which is so bad that the parallel implementation runs slower than a serial one from which it was developed due to gross communication overheads.

Process mappings to processors fall into two broad categories: functional decomposition and data decomposition.

In *functional decomposition*, the various functions of the algorithm are distributed amongst the processors and data flow from processor to processor. An example of this is a pipeline with, say, the front-end process on the first processor, followed by various subsequent stages on the other processors. This decomposition can be costly in terms of communications time and it is difficult to balance the processing load. Sometimes it is necessary if, for example, the processors in the system are different (hybrid architecture).

In *data decomposition*, instead of partitioning the algorithm, we partition the data into portions which are processed on different processors. Load balancing is fairly automatic. Perfect balancing is achieved if the processing is data independent. In this case we have what is known as a Task Farm. The remaining problem is to choose the size of the data portions. It should be chosen to keep all of the processors busy all of the time (as near as is possible). For these reasons, data decomposition was employed in both methods described in this paper.

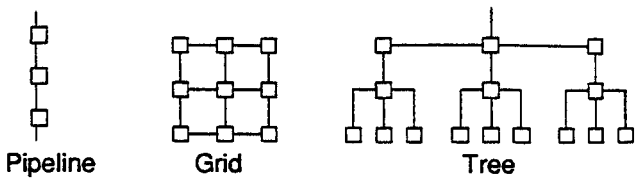

Fig. 3: Traditional, simple processor network topologies

4.1 Software routers and network topologies

All the programs were written in C++. This enabled us to employ object orientated techniques. The message passing was greatly assisted by the use of CHIMP (Common High-level Interface for Message Passing). CHIMP is a topology and processor independent routing harness. Thus it was possible for these implementations to be developed under the familiar UNIX environment on workstations before finally being ported to a network of transputers.

4.1.1 Processor network topology

As parallel computers' processors become faster, more densely interconnected and more numerous, the topology of the processor network becomes less important. In theory the best parallel computer has an infinite number of processors, all of which have a direct link to all other processors with infinite bandwidth. Any given application would run on a small 'corner' of the network. However, today's MIMD (multiple-instruction, multiple-data) machines have a few processors sparsely interconnected with low bandwidth. Although the technology is improving, we still need to consider the network topology.

In the past message routing was the responsibility of the programmer and so simple processor networks (Fig. 3) were used because of their associated simple routing algorithms. Nowadays however, complex topologies (Fig. 4) can be employed along with a topology-independent routing harness such as CHIMP. Recent research has shown that the best performing topologies can be complex or even random [5]. The 'greedy' ring topology was employed here since the aim was to minimise communications distance between the data source and all of the processors, whilst maximising the available bandwidth. If, for example, a tree topology were used, there are no alternative routes for messages which can lead to overused links.

The hardware used for this implementation is a network of T800 transputers [6]. Each processor has four bi-directional links by which they can be interconnected. The network is totally reconfigurable with the constraint that one of the links must connect to the host computer for file access.

4.1.2 Communications

Overall, the efficiency of a parallel system is improved by minimising the amount of communication. The way to

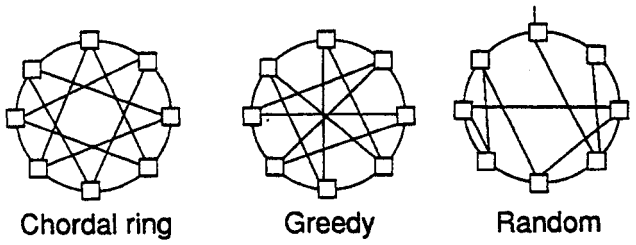

Fig. 4: Complex processor network topologies

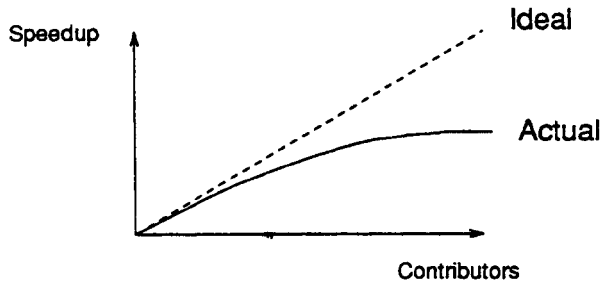

Fig. 5: Amdahl's Law

minimise communication is to do as much work as possible on a piece of data before sending it elsewhere. Thus, for example, the philosophy for a pipeline of processes should be to have all the pipeline of processes available on *every* processor being used. In this way, pieces of data can be 'farmed out' to worker processes by a master process.

In the case of Transputers, the bandwidth of each link is constant at 20 Mbit/s. Each message sent down a link involves some duplex handshaking. Thus the larger the message the better.

When a parallel program is 'communications bound' the only ways of improving efficiency are:

- Add more links for extra bandwidth;
- Modify the communication strategy to reduce the number of messages.

The programmer can benefit from intelligent routers which will keep a table of all shortest routes between processors. Very intelligent ones can adapt their choice of route to avoid heavily loaded links.

5 ADOPTED MAPPINGS

Amdahl's law [3] states that the inherently sequential parts of a program dictate the maximum possible speedup achievable by running parts of the program in parallel (Fig. 5).

5.1 Method 1

To aid description of the parallel processes and their interaction an analogy is drawn between VQ codebook generation and the production of an academic anthology. There is a unique process known as the *Editor* which has global overview of the codebook generation and collates results whilst not actually generating any results itself. There are several identical processes known as *Contributors* which each have a disjoint subset of the training vectors and are able produce partial results which they communicate with the other processes (Fig. 6). Each process runs on a seperate processor.

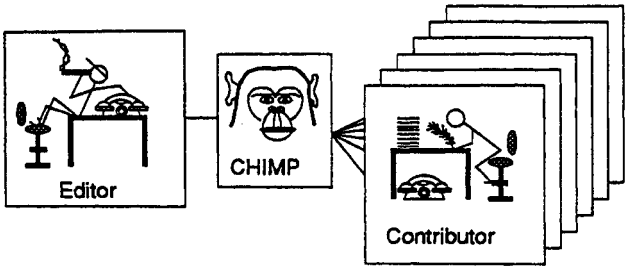

Fig. 6: Method 1—Editor/Contributor mapping

In order to ensure that the load is evenly balanced, it is only necessary to ensure that each Contributor has the same number of training vectors. The reclassification process involves all the Contributors allocating all their training vectors to one of the clusters. Therefore if they each have the same number of training vectors, the load is perfectly balanced.

5.1.1 Algorithmic modifications

When clusters are split it is necessary to choose the centroids of the two new clusters. The best results have in the past been achieved by a minimax criterion [7]. However, this involves knowing all the member training vectors for the cluster being split. As an alternative, it was decided to employ the 'farthest neighbour' technique [8]. All Contributors know which of their training vectors is most distant from the centroid. Of these, the Editor selects the globally most distant as a new cluster centroid and the old centroid is kept as the other. Thus, for each cluster, the Editor can decide which Contributor has the most distant training vector and tell it to split the cluster.

This method is attractive because no synchronisation is necessary during the computationally intensive parts of the algorithm (i.e. training vector reclassification and the calculation of the cluster centroids). Provided each Contributor runs on a separate processor with an equal number of training vectors, the computational load is automatically balanced. To maximise the parallelisation in accordance with Amdahl's law the work of the Editor (which is a sequential bottleneck) must be minimised. In Method 1 the Editor process performs less than 1% of the total processing which means that it can run on the same processor as one of the Contributors without affecting the load balancing significantly.

5.2 Method 2

In Method 1 each Contributor had responsibility for its own subset of training vectors over the global set of clusters. The second method distributes the clusters rather than the training vectors. Each Contributor holds locally as many training vectors as memory constraints will allow. Thus their subsets are no longer disjoint and in the best case all of the training vectors are held at every Contributor.

The cluster centroids are broadcast to all Contributors but individuals have the responsibility of working on a disjoint subset of the clusters. It may be necessary for Contributors to occasionally exchange a few of the training vectors as their 'ownership' changes if not all are held by each. Thus although each Contributor holds many or most of the training vectors, they only own some of them at any given time.

The larger memory requirements make this second method better suited to SIMD (single-instruction, single-data) machines. It would perform best of all with shared memory which is often used in SIMD machines.

6 CONCLUSIONS

It can be argued that the speed of execution of the k-means is not particularly important when using the results as static codebooks for speech recognition systems. However, in the research environment it is absolutely necessary to be able to experiment with various codebooks so as to attain exactly what makes the best ones. Without fast codebook generation, this is extremely tedious.

VQ codebooks generated by the algorithms described in this paper have successfully been employed in a semi-continuous hidden Markov Model-based speaker verification system [9]. They have also been used in our parallel speech recogniser [10]. Our interest has been in affordable MIMD machines but there is no reason why Method 2 could not be implemented on a more expensive SIMD and more massive speedups achieved.

7 ACKNOWLEDGEMENTS

The research described in this paper was carried out using the facilities at the Centre for Speech Technology Reasearch, University of Edinburgh, Edinburgh, Scotland, UK. The authors wish to acknowledge The Edinburgh Parallel Computing Centre for providing CHIMP. Funding was supplied by Cairntech Ltd. and the UK Government's DTI as part of their Parallel Applications Programme.

REFERENCES

[1] L. Rabiner, S. Levinson, A. Rosenberg, and J. Wilpon, "Speaker-independent recognition of isolated words using clustering techniques," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-27, no. 4, pp. 336–49, August 1979.

[2] Y. Linde, A. Buzo, and R.M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. on Communications*, vol. COM-28, no. 1, pp. 84–95, January 1980.

[3] G. Amdahl, "The validity of the single processor approach to achieving large scale computing capabilities," In *AFIPS conference proceedings, Spring Joint Computing Conference*, volume 30, pp. 483–485, 1967.

[4] S. Bokhari, "On the Mapping Problem," *IEEE Transactions on Computers*, vol. C-30, no. 3, pp. 207 – 214, March 1981.

[5] D.M.N. Prior, N.J. Radcliffe, M.G. Norman, and L.J. Clarke, "What Price Regularity?," *Concurrency: Practice and Experience*, vol. 2, no. 1, pp. 55–78, March 1990.

[6] M.B. Sandler, L. Hayat, L. Costa, and A. Naqvi, "A Comparative Evaluation of DSPs, Microprocessors and the Transputer for Image Processing," *in IEEE Proceedings ICASSP*, vol. 3, pp. 1532 – 1535, 1989.

[7] S. Levinson, L. Rabiner, A. Rosenberg, and J. Wilpon, "Interactive clustering techniques for selecting speaker-independent reference templates for isolated word recognition," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-27, no. 2, April 1979.

[8] T.W. Parsons, *Voice and speech processing*, McGraw-Hill, NY, 1987.

[9] M.E. Forsyth, A.M. Sutherland, J.A. Elliott, and M.A. Jack, "HMM speaker verification with sparse training data on telephone quality speech," In *Proceedings of the Fourth Australian International Conference on Speech Science and Technology (SST-92), Brisbane*, pp. 67–72, December 1992.

[10] J.A. Elliott, F.R. McInnes, N.W. Ramsey, and M.A. Jack, "A parallel processing keyword recogniser for police national computer enquiries," In *Proc. European Conference on Speech Communication and Technology*, 1993.