Parallelization of automatic speech recognition

Ould Mohamed El Hadj Yahya, Nathalie Revol

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 5668

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

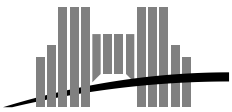
SPI

Parallelization of automatic speech recognition

Yahya Ould Mohamed El Hadj
Nathalie Revol

January 2001

Research Report N° 2001-02



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

INRIA

Parallelization of automatic speech recognition

Yahya Ould Mohamed El Hadj
Nathalie Revol

January 2001

Abstract

The automatic recognition of spoken words is increasingly common, for dictaphone applications, telephone services or the command of various devices by disabled persons. In the latter case, a high recognition rate is expected on a vocabulary of small to medium size. To achieve this goal, the model must be refined. Thus, both the training stage and the recognition stage for such applications can be very time consuming and occasional re-training may happen. Its parallelization is thus worth considering. In this paper we present firstly the models we use: the classical hidden Markov model and another model that takes into account the prosody of speech, namely the centisecond two-level hidden Markov model introduced by Meziane [10]. Then two parallelization strategies are detailed: the first one simply shares the vocabulary among the processors, the second one also distributes the model. Experimental results highlight the need for a finer load-balancing: an *a priori* load estimation is presented and is used to statically balance the computational load between the processors. Further experiments have been conducted and exhibit efficiencies higher than 65% on an architecture composed of 12 Pentium Pro interconnected via Myrinet. Directions for improving further the parallelization are given.

Keywords: automatic speech recognition - hidden Markov model - centisecond two-level hidden Markov model - parallelization

Résumé

La reconnaissance automatique de la parole est utilisée de plus en plus couramment, par exemple pour la commande de divers appareils par des handicapés. Cette application nécessite un taux de reconnaissance très élevé pour un vocabulaire de taille petite à moyenne et le modèle doit donc être particulièrement précis. Les phases d'apprentissage et de reconnaissance sont dans ce cas très gourmandes en temps et des ré-apprentissages peuvent se produire. Le besoin de paralléliser cette application se fait alors sentir. Dans ce rapport, nous commençons par présenter les modèles que nous utilisons, à savoir le modèle de Markov caché classique et le modèle de Markov caché à deux niveaux centiseconde introduit par Meziane [10], qui permet de tenir compte de la prosodie de la parole. Nous présentons ensuite deux stratégies de parallélisation, la plus simple consistant à partager les mots du vocabulaire entre les processeurs et l'autre répartissant également le modèle. Les résultats expérimentaux mettent en évidence la nécessité d'utiliser une répartition de la charge plus élaborée. Nous définissons des formules d'estimation *a priori* de la charge et les utilisons afin de distribuer statiquement la charge de calcul. Les expériences correspondantes présentent des efficacités supérieures à 65% sur une architecture composée de 12 Pentium Pro interconnectés par un réseau Myrinet. Nous indiquons en conclusion quelques pistes en vue d'améliorer cette parallélisation.

Mots-clés: reconnaissance automatique de la parole - modèle de Markov caché - modèle de Markov caché à deux niveaux centiseconde - parallélisation

1 Introduction

In the domain of speech recognition, different kinds of applications can be found, each differing in its use and in the expected performances. The three main discriminating criteria concern the size of the vocabulary which can be small or large, the facts that words are connected in their utterance (continuous speech) or isolated (there are pauses between words) and the number of intended speakers (only one speaker or multiple speakers).

The large vocabulary continuous speech recognition tries to mimic the human capabilities, it is intended for dictation applications for instance. The difficulties are firstly to memorize and to recognize – possibly without a single corresponding sample in the database – a large number of words, typically over 20 kwords, secondly to be able to split the input into separate words and finally to identify syntactically correct and semantically meaningful sentences. The expected error rate is therefore quite high: above 8% [7] or 5% [24], but an error can be corrected afterwards without implying any serious consequence. Furthermore, such a dictaphone can be tuned to recognize a single speaker and this enables it to reduce the error rate down to 3-5%.

Another kind of application deals with medium to large vocabulary but without the further assumption of continuous speech. Such applications are for instance interactive telephone voice services on specialized domains (cinemas audio services or consulting and giving order for a bank account for instance). For these applications the size of the vocabulary is again a problem. The problem to identify the words' frontiers doesn't hold any more, but the independence regarding the speaker is really an issue. The language restrictions on the syntax and semantic constraints may subsist in the case of a dictaphone application (the early dictaphones imposed a pause between words) or disappear in the case of a telephone service. In the latter case the problem may be to wrongly identify a spoken word as "out of vocabulary" when it is not, and to deal with various transmission channels (mobile telephones for instance) or extra noise (running car etc [13]). The problem is also to keep a high recognition rate: even if an error does not lead to serious problem, too many errors discourage the potential users.

A last kind of automatic speech recognition system deals with unconnected words from a small to medium size vocabulary. The possible applications are the command of various devices by disabled persons or dialing telephone numbers via an audio command: the recognition system has to be speaker-independent and its key issue is to provide a very high recognition rate in order to be of real use: an error rate of a few percents is required. This is the problem we are dealing with in this paper.

The model has to be more precise than for the other cases in order to achieve this target. However, such a refinement can lead to algorithms which are not only time consuming but also memory demanding. To sum up, our aim is to speed up both the processing time and the storage capacity for isolated word systems which are designed to recognize very accurately a small to medium size vocabulary, independently of the speaker. The main issue for this kind of application is not the vocabulary size but the quality of the recognition. No particular assumption is made on the vocabulary, which can for instance contain words having close pronunciations. The parallelization has to preserve the sequential recognition rate.

In this paper the decomposition of words into smaller units, namely pseudo-diphones, and the modelling of these units by hidden Markov models (abbreviated as HMM) or by centisecond two-level hidden Markov models (abbreviated as cTLHMM) will be detailed in section 2. HMM are the most widely spread models used in speech recognition, due to their established efficiency. However they do not take into account the duration of each phone (prosody) and cTLHMM have been introduced to handle the prosodic aspects. In section 3 the need of speeding up the speech recognition task and in particular the training stage, even in the case of isolated words with medium vocabulary, is justified: the training stage can be very long even on such applications and if there is a need for re-training (to tune the system for a new speaker of for a new sound device such as the microphone for instance), this re-training has to be very fast. Two parallelization strategies are then presented; the first one distributes the computations among the processors but replicates the model; the second one aims at distributing the model as well as the computations among the processors. It then appears from experiments that load balancing is an important issue towards performances: section 4 explains our *a priori* load estimation and our static distribution strategy and gives thoroughly commented experimental results: we got speed-ups of 7.6 on 10 processors. We conclude by emphasizing our contribution and by discussing other possibly promising approaches for the parallelization of this application.

2 HMM and cTLHMM

2.1 Acoustic modelling

In our experiments, the speech is observed through the telephone, which implies that the observed frequencies belong to the interval [50Hz, 3.3kHz]. The speech waveform is sampled and digitalized by standard devices available in many workstations and advanced PCs. It is then divided into blocks of 32ms, where blocks are overlapped. A pre-accentuation filter is applied to equalize low and high tones, the latter having usually less energy, then a Hamming window function is applied to each block. A Fourier transform is thus performed on each block, giving a frequency spectrum of the signal. A Mel filter is applied in order to mimic the human ear which perceives more accurately bass tones than high-pitched ones. A logarithm is applied to the resulting spectrum, its effect being to make the distribution Gaussian (as this will be useful later, cf. §2.2). Finally an inverse Fourier transform is done and the 8 most significant components (the 8 first ones) are kept, they are called Mel frequency cepstral coefficients (MFCC). The total signal energy in the window is the 9th and last component of our acoustic vectors.

A spoken word is thus described by a sequence $Y = (y_t)_{1 \leq t \leq T}$ of acoustic vectors y_t each belonging to $\mathbb{R}^9$. The speech recognition task consists in determining the word $\hat{W}$ having the maximal likelihood given the observed acoustic sequence Y, *i.e.*

$$\hat{W} = \arg\max_W P(W|Y) = \arg\max_W \frac{P(W)P(Y|W)}{P(Y)}.$$

$P(W)$ is the probability that the word W is pronounced, it depends only on the vocabulary and not on Y; for the sake of simplicity and without further assumption on the vocabulary it is assumed to be uniform on the vocabulary. $P(Y|W)$ is the probability to observe Y given some specified word W is pronounced, it depends on the acoustic model (cf. below). To determine $\hat{W}$, the quantity $P(Y|W)$ is computed for every word W in the vocabulary and $\hat{W}$ is the word leading to the maximal value.

The construction of the acoustic model could rely on the construction of an acoustic model for each word in the vocabulary and on the union of these models, but it would take too much place in memory. In practice, each word is split into smaller phonetic units that can be shared by several words. This motivates the segmentation of the signal in small blocks, in order to match the phonetic units, instead of a global processing which would imply a global analysis. In this work, the phonetic units are phones and also diphones, in order to take into account the coarticulation phenomena (*i.e.* the fact that the pronunciation of a phone depends on the neighbour phones): this set of phones and diphones is called *pseudo-diphones* [2].

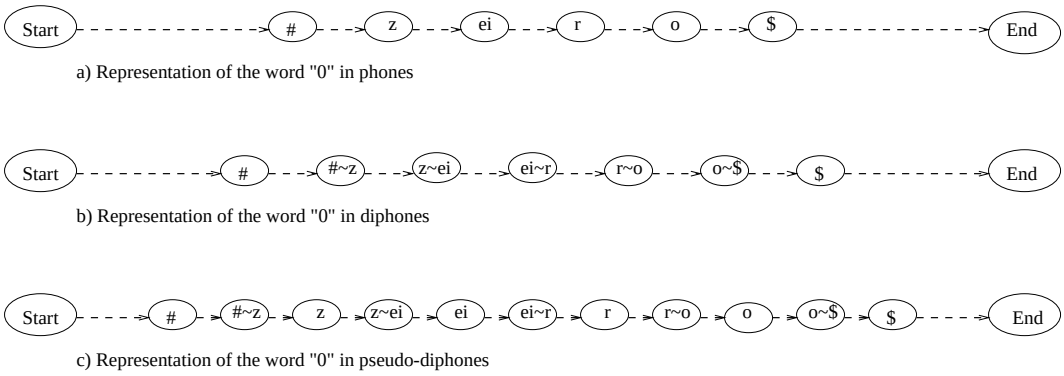


Figure 1: Example of the various phonetic units: phones, diphones and pseudo-diphones, for the word “zero”.

2.2 Hidden Markov models or HMM

Definition [17]

For each pseudo-diphone, a small network is built. The number of nodes is typically between 3 and 5.

For each such network, an entry node and an exit node are added, enabling to easily join two networks by merging the exit node of the first network with the entry node of the second one. The network corresponding to a word is obtained by joining the networks associated to its pseudo-diphones and the global network for the vocabulary is obtained by joining the networks of its words. With nodes being called states and edges being called transitions, it is a finite state automaton.

At a discrete instant t, it goes from state i to state j using the transition (i,j), where j can be equal to i, and generates an acoustic vector y_t. The choice of this (i,j) transition starting from state i is random with probability $a_{i,j}$; the acoustic vector y_t is also generated with probability $b_{i,j}(y_t)$ (this probability can depend on the state i or on the transition (i,j), in this paper it depends on the transition in order to enable a finer modelling when needed, however some $b_{i,j}$ may be put in common by several transitions if there is no need to distinguish between them).

This is named a Markov model since the probability of going into state j does only depend on the previous state i via the probability $a_{i,j}$ but not on the preceding ones, which means that the sequence of visited states is a first-order Markov chain. It is called a hidden Markov model since the states i cannot be directly observed: they are hidden, indeed only the corresponding acoustic vectors y_t are observed.

In a hidden Markov model M with parameters $(a_{i,j})$ and $(b_{i,j}())$, the joint probability of a sequence of acoustic vectors $Y = (y_t)_{1 \leq t \leq T}$ and of a sequence of states $X = (x_t)_{0 \leq t \leq T+1}$ with x_0 the entry state, x_{T+1} the exit state and x_t one of the possible other states for $1 \leq t \leq T$ is

$$P(Y, X|M) = a_{x_0,x_1} \prod_{t=1}^{T} a_{x_t,x_{t+1}} b_{x_t,x_{t+1}}(y_t).$$

Since we assume that the entry state is unique as well as the exit state, the probability of starting in x_0 and ending in x_{T+1} is 1 and does not appear in this formula. The probability to observe the sequence Y of acoustic vectors in the model M, $P(Y|M)$, is the sum over every possible sequence X of states of $P(Y, X|M)$. In order to simplify this computation, an usual approximation consists in replacing $P(Y|M)$ by the maximum over every possible sequence X of states of $P(Y, X|M)$. This is computed by Viterbi algorithm, which is a classical algorithm in dynamic programming.

Determination of the parameters $a_{i,j}$ and $b_{i,j}()$: training stage

This is called the training stage. It requires, for each word of the vocabulary, a set of utterances: since our vocabulary is small- to medium-sized, we do not suffer from data sparsity and have several utterances for each word.

The determination of the $a_{i,j}$ and $b_{i,j}()$ is done by an iterative procedure. At a given step, the current model is denoted by M. For each utterance Y^r in the data base, Y^r is a set of acoustic vectors and we denote by X^r the corresponding sequence of states:

$$X^r = \arg\max_X P(Y^r, X|M)$$

then we determine a new model M' which maximizes among every possible model M'' the quantity $\prod_r P(Y^r, X^r|M'')$. For numerical reasons, namely to avoid underflows, the logarithm of this quantity is considered. The problem is thus to find

$$M' = \arg\max_{M''} \sum_r \log P(Y^r, X^r|M'').$$

The Lagrangian for this problem is formed from this expression and the constraints stating that the unknown parameters represent probabilities (for instance, $\forall i, \sum_j a_{i,j} = 1$). At an extremal point, all partial derivatives of the Lagrangian with respect to any variable must cancel, which leads to the formulae for the parameters $a'_{i,j}$ and $b'_{i,j}()$ of the new model M'; for instance

$$a'_{i,j} = \frac{\sum_r \text{nb of times the transition } (i,j) \text{is used in } X^r}{\sum_r \text{nb of times the state } i \text{ appears in } X^r}.$$

It can be shown that this iterative algorithm converges towards a local maximum of the function of M

$$\sum_r \log P(Y^r, X^r|M).$$

If the function to be maximized is $\sum_r \log P(Y^r|M)$, an algorithm is also available: Baum-Welch procedure.

Recognition stage

A given sequence Y of acoustic vectors is observed : $Y = (y_t)_{1 \leq t \leq T}$ and the recognized word is the word corresponding to the sequence of states $X^* = \arg\max_X P(Y, X|M)$.

Parameters

The $(a_{i,j})_j$ correspond to the statistical use of the transitions exiting state i, initially they are supposed to be uniform or they are given values obtained from a preliminary statistical study.

The $b_{i,j} : \mathbb{R}^9 \to [0,1]$ are chosen to be Gaussian distributions (cf. the logarithm applied in the frequency domain) $\mathcal{N}(\mu_{i,j}, \Sigma_{i,j})$ where the covariance matrix $\Sigma_{i,j}$ is assumed to be diagonal, *i.e.* the components of an acoustic vector are assumed to be independent.

2.3 Centisecond two-level hidden Markov models (cTLHMM)

Why integrating the phones' duration?

Specialists in phonetics have put in evidence that phones can be distinguished by their duration as well as by their acoustic features. Each phone exhibits a duration inherent to the localization and the mode of its articulation [22], and this cannot be compressed under a minimal duration nor last beyond a maximal duration [23]. The duration also contains informations on the linguistic context of the phone [15]. Furthermore, duration can enable to distinguish between two acoustically close sentences; a convincing example is given by the two following sentences: "it was the topic of the year" and "it was the top pick of the year". When the duration of the occlusive /p/ in "topic" increases, one recognizes the second sentence instead of the first one [12]. See also [21, 4] for a discussion of the introduction of phones' duration into speech recognition devices.

Proposed solutions

The duration is implicitly modelled by a HMM, thanks to the transition probabilities: the probability of staying during d units of time in state i is $a_{i,i}^d(1 - a_{i,i})$; however, they correspond to geometric distributions for the durations, whereas the experimentally observed distributions are well fitted by Gamma distributions (cf. [3]).

Various solutions are proposed in order to integrate the duration into the model. A solution proposed by [18] is the extended states HMM: each state of the HMM is replaced by a sub-network modelling the duration. The problems are then the determination of a good topology for these sub-networks and also the increase in size of the training database since there are more parameters to estimate.

Tuerk and Young [20] devised a "two-speed HMM" (the denomination is ours): each HMM corresponding to a logical phonetic unit is duplicated and one copy corresponds to a slow pronunciation whereas the other corresponds to a fast one. The problem of determining a good topology disappears but not the requirement for a larger training database.

Semi-HMM have been proposed in [9]: the sojourn duration in a state is explicitly modelled via a probability describing it. It is called semi-HMM because the Markov property holds only between consecutive different states but not when the state remains the same. With this model it is mandatory to adapt the training and the recognition algorithms and the new algorithms are more time-consuming than the classical ones (Baum-Welch or based on Viterbi).

Segmental TLHMM have been proposed by Suaudeau [19], it relies on a *a priori* segmentation of the signal into phones at the signal processing level, *i.e.* the blocks have a variable length, corresponding to the length of the phone; however, this segmentation is not reliable.

Centisecond two-level HMM

The idea for proposing cTLHMM was to avoid the unreliable segmentation at the acoustic level in order to deal with the duration afterwards and thus to be able to correct mistakes since it is now done at the same stage. The acoustic phase is thus the classical one introduced in §2.1. A centisecond TLHMM [11] is a hierarchical construction above a classical HMM (hence the "two-level" appearing in its name) using blocks of fixed size, of the order of magnitude of the centisecond. The HMM is still used as the acoustic model and a phonetic level is added. The link between these two levels is an application which gives, for each state in the sequence of states, the corresponding phonetic state as well as the entry date in this phonetic state. Several consecutive states at the acoustic level can thus be associated to the same phonetic unit with the same entry date.

Let's state this more mathematically. $Y = (y_t)_{1 \leq t \leq T}$ denotes the sequence of observed acoustic vectors. The acoustic HMM is characterized by the transition probabilities $a_{i,j}$ and by the probability distributions $b_{i,j}()$ where $b_{i,j}(y_t)$ is the probability that y_t is observed during the transition from state i to state j. $X = (x_t)_{0 \leq t \leq T+1}$ denotes a sequence of states in this acoustic HMM. At the phonetic level, $\Omega = (\omega_\theta)_{1 \leq \theta \leq \Theta}$ is the process representing the corresponding sequence of phonetic units. The number Θ denotes the total number of phonetic units in the sequence and $\omega_\theta = (\varphi_\theta, \psi_\theta)$ with φ_θ the θ-th phonetic unit in the sequence and ψ_θ the entry time in φ_θ or equivalently the time index of the first state in X which is in the phonetic unit φ_θ $(1 \leq \psi_\theta \leq T)$.

The sojourn duration in the phonetic unit φ_θ is distributed according to a probability depending on φ_θ only: $\rho_{\varphi_\theta}(d)$ is the probability of staying d units of time in φ_θ. Figure 2 helps to get a more visual understanding of this definition.

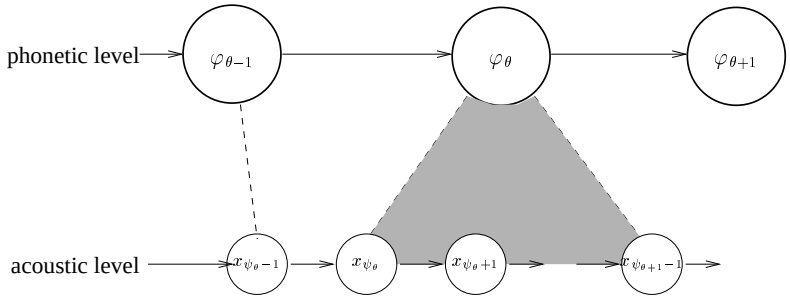


Figure 2: The two levels in a cTLHMM.

The joint probability of the observed sequence Y of acoustic vectors and the hidden sequence X of acoustic states is thus

$$P(Y, X|M) = a_{x_0,x_1} \times \prod_{t=1}^{T} a_{x_t,x_{t+1}} b_{x_t,x_{t+1}}(y_t) \times \prod_{\theta=1}^{\Theta} \rho_{\varphi_\theta}(d_\theta)$$

where $d_\theta = \psi_{\theta+1} - \psi_\theta$ is the sojourn duration in the θ-th phonetic unit.

The algorithm computing this quantity is a slight adaptation of Viterbi algorithm using 3 indices instead of 2 for the classical one (cf. §4.1); the extra index stands for the sojourn duration in a phonetic unit and belongs to the interval $\{1, d_{\max}\}$ where $d_{\max}$ is the maximal sojourn duration in any phonetic unit. The cost of this modified Viterbi algorithm is thus $d_{\max}$ times the cost of the usual one, both in time and memory. Since the value of $d_{\max}$ is 15 in our experiments, this is a reasonable overhead.

The training stage determines the parameters for the ρ_φ laws in addition to the other parameters. In this paper, the ρ_φ are taken to be Gaussian, since reestimation is easier with Gaussian laws than with Gamma laws. The training phase is split in three steps aiming at reducing the computational cost (by optimizing a classical HMM first) and accelerating the convergence. The first step is the training presented for the HMM, and the duration parameters are not taken into account. Then the duration parameters are initialized with values corresponding to the recognition of the utterances of the training database. Lastly, a training phase

similar to that for HMM and including the duration parameters is performed. Experiments indicate that good results are obtained without increasing the number of utterances in the training database [11]: the recognition rate of error drops from 4.8% with HMM to 3.2% with cTLHMM on the training database and from 6.1% to 3.7% on the test database.

3 Parallelization

3.1 Need for accelerating the speech recognition device

A first and obvious motivation for accelerating the speech recognition device is the quest for real-time and low error-rate system.

The training stage alone can benefit from parallelization, especially with the required low error-rate in our target applications. Indeed, the training stage can take a long time with a medium-size vocabulary, a large training database and words having close pronunciations for instance. Meziane [10] reports experiments during 2 weeks on a workstation: the vocabulary was composed of the 500 most frequent words of the French language. Furthermore, re-training or rather further training iterations can be useful when there is a need to adapt the system to a new speaker, to a new audio device such as a microphone or to a new transmission channel such as a mobile telephone. In these cases, it has to be very quick since the speech recognition system is already in operating mode and users won't be willing to wait for a too long time.

Our target parallel computer is simply a pile of PC, since it is the most widely available parallel machine at the lowest price. The interconnection network is Myrinet (1.28Gb/s) and the communication protocol is BIP [1]. Since the target vocabulary has a limited size, it is possible to build and store explicitly the corresponding HMM or cTLHMM in memory.

3.2 Some existing algorithms

To our knowledge, only few parallel algorithms have been proposed in the literature. The main problem consists in finding independent computations in Viterbi algorithm, which can be stated as follows:

```
for t = 1 to T do
  for j = 1 to nb_of_states do
    delta(t,j) = max delta(t-1,i)*a(i,j)*b(i,j)(y(t))  -- max over every possible i
```

it exhibits strong dependencies in the t loop.

A proposal could thus be to split the j loop into several parallel computations. It can be found in [16]: the HMM is implicit and its set of states is built when needed; the states are distributed cyclically among processors and each computational task handles the transitions exiting its states. However this algorithm implies, on a shared-memory parallel computer, a clever locking algorithm for the intensively accessed hash table. It is difficult to adapt it for a distributed memory computer since it would involve too many communications between processors (intuitively due to the transitions between states), *i.e.* too many synchronizations which are known to heavily deteriorate the performances.

Another proposal of the same kind [7] consists in distributing the processing of states among processors in a master-slaves way, however "worker processors each hold copies of a pool of 1.954 tri-state models". Care is taken to communicate only re-estimated probabilities at each iteration and further parallelism is obtained at the slaves level by multithreaded programming.

Another solution consists in breaking the dependencies in the t loop by using old values of `delta(t-1,i)` for the computation of `delta(t,j)` and by iterating this process: at the p-th step,

$$\delta^{(p)}(t,j) = \max_i \delta^{(p-1)}(t-1,i) a_{i,j} b_{i,j}(y_t)$$

This is a simplified summary of [14]. Unfortunately no experiments conducted on a parallel computer are presented. Furthermore, this method increases the error rate by about 0.4% in the reported sequential experiments.

3.3 Parallelization strategy for the training stage

We have already mentioned that the training stage can be very long or that some quick training iterations may have to be performed from times to times and thus that the training stage deserves parallelization. Our target parallel computer is a distributed memory architecture with possibly a large number of processors.

The principle of this parallelization consists in splitting the vocabulary among the processors, *i.e.* in parallelizing the loop on words and utterances implicitly enclosing Viterbi algorithm given in §3.2. Each processor is in charge of certain words and it has to train on the set of utterances of its own words. In this strategy, the network (HMM or cTLHMM) is copied onto the memory of each processor and thus we will refer to it as *replication strategy* in our experiments. Since a processor handles all utterances of a limited number of words, it will determine optimal paths X^r for utterances Y^r that use a limited amount of states and transitions. Hence only probabilities related to these states and transitions will be broadcast to the other processors, *i.e.* this strategy aims at limiting the volume of communications. Another advantage of this strategy consists in the limited modifications of the sequential program. The main drawback is that the program does not benefit from the increase of the memory size.

The corresponding parallel iteration of the training is the following:

- each processor computes, for each utterance Y^r of a word belonging to its set of words, $P(Y^r|M)$ or rather $P(Y^r, X^r|M)$;
- each processor counts the number of uses of each state and each transition at each instant, *i.e.* its partial sums for the sums appearing in the reestimation formulae;
- each processor broadcasts its partial sums to the other processors (or reductions can be performed to compute the sums of these partial sums) and receives the corresponding information from the other processors;
- each processor updates the network parameters.

In [6], this parallelization scheme is used. The overlapping of communications by computations is emphasized: the processing of words is split into several parts and communications occur at the end of each part, in order to be overlapped by the processing of the following part. In our experiments, communications take a negligible part of the total time and thus no particular effort is made to overlap them apart from the obvious ones such as performing some local updates simultaneously with the communications.

This parallelization is dedicated to the training stage and cannot be used for the recognition stage in general: indeed, each processor looks for the optimal path in its network, which is the global network. The only possibility for parallel recognition occurs when several words have to be recognized simultaneously.

3.4 Parallelization strategy for the training and recognition stages

This strategy consists in distributing not only the vocabulary but also the network among the processors, in order to circumvent the drawback of the previous parallelization strategy and to enable parallel recognition as well.

The principle of this parallelization is the following: each processor gets a subset of the vocabulary and a network (HMM or cTLHMM) corresponding to this sub-vocabulary. This means that the model of a phonetic unit is not shared between two processors or even between two words on the same processor, or in other words that the probabilities (*i.e.* the distributions as well as the parameters) will be different if they appear on different words or processors: they are duplicated but not replicated as it would mean that two copies are identical. In our experiments this strategy will be referred to as *distribution strategy*. The only phonetic units having the same replicated model with the same parameters on every processor are the initial and final silences.

The advantages of this approach are that the memory of the whole parallel computer is better employed and also, thanks to the probabilities duplication, that less communications are required, namely only the communications for the parameters of the initial and final silences. The drawback has already been mentioned for other models: an increase in the number of parameters implies an increase of the size of the training

database. Experimentally we did not observe a loss but on the contrary an improvement of 3.5% of the error-rate, with a vocabulary of 20 words and with the same training database.

The following parallel algorithm implements this distributed strategy for the central training iteration:

- each processor computes, for each utterance Y^r of a word belonging to its set of words, $P(Y^r|M)$ or rather $P(Y^r, X^r|M)$;
- each processor broadcasts to the other processors its partial sums for the parameters of the models of silence and receives the corresponding information from the other processors;
- each processor updates the network parameters.

With this strategy, the recognition can also be done in parallel:

- each processor computes the optimal path corresponding to the input word in its local network and its probability;
- the maximal probability among all processors is retained and the answer is the word associated to the corresponding path.

If only the recognition had to be parallelized, then it could be possible to replicate the models for the different phonetic units, *i.e.* to use the same probabilities (same distributions and same parameters) for each occurrence of the same phonetic unit.

3.5 First experimental results

Our experiments use vocabularies of 20 and 50 words (the first numbers, in French), each has been uttered on the telephone twice by each of the 10 speakers for the training database and once for the recognition tests. The target vocabulary has 500 words, but since the experimental programs have been firstly developed in PVM for the Telmat TN310, which is quite an old machine (32 T9000 Transputers with 8Mb memory each), it was impossible to handle the complete vocabulary and we had to restrict it to 50 words. The program has then been ported to MPI and run on another machine. The parallel computer on which these experiments have been conducted is composed of 12 Intel Pentium Pro 200MHz interconnected via Myrinet. The message passing library is MPI, built on top of BIP which is specially tuned for this network. The words are randomly distributed to the processors, the only criterion – without any prior assumption or knowledge on the words – is that the number of words is approximately the same (up to 1) on each processor. This load balancing is performed *statically*, *i.e.* it occurs during an initialization stage and is never modified during the computations.

The experimental times are shown on figure 3. The distribution strategy uses a model different from the one used by the replication strategy and has more parameters to determine; the execution times are thus not fully comparable. However, there is a gain of time in favour of the distribution strategy when the number of processors increases: an explanation of this phenomenon is given at the end of this section.

The experimental efficiencies are shown on figure 4. The definition of the efficiency is the ratio $\frac{t_{seq}}{p.t_{//}}$ where p is the number of processors, t_{seq} the time of a sequential execution and $t_{//}$ the time of a parallel execution on p processors. They are not dramatic: an ideal efficiency would be 100%.

The execution times of the fastest processor and the slowest processor are reported in the table below, along with the relative difference ($\frac{t_{slow}-t_{fast}}{t_{slow}}$): these values put into light an imbalance of the working load. This imbalance is greater for the replication strategy than for the distribution one: this may explain the better times and efficiencies obtained with the latter one. More precisely, the computation of the optimal path depends tightly on the processed word and intuitively on its length. Hence, what these experiments strongly suggest is the need of a better load balancing strategy. In the next section an *a priori* estimation of the amount of computation implied by each word is given, then a static load-balancing algorithm is presented and experimental results illustrate the improvements.

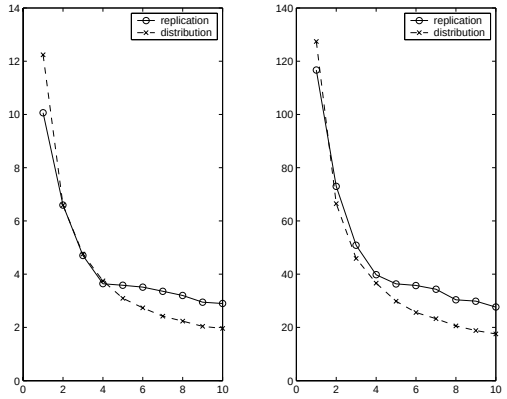


Figure 3: Times in s. versus the number of processors for the replication and the distribution strategies, with HMM (left) and cTLHMM (right).

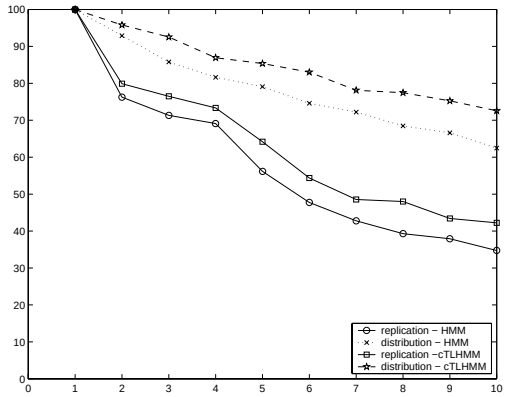


Figure 4: Efficiencies versus the number of processors for the replication and the distribution strategies.

4 Load-balancing: algorithm and experimentations

4.1 A priori estimation of the load

Since the number of words or the number of pronunciations to be processed is not a reliable estimation of the actual workload, a finer *a priori* estimation is devised. Intuitively, the lengths of the pronunciations are related to the complexity of their processing. This is confirmed by the following analysis of Viterbi algorithm.

HMM

Let's denote by N the number of states. For a given pronunciation with T observed acoustic vectors, Viterbi algorithm is:

`for` t `= 1 to` T `do`
  `for` j `= 1 to` N `do`
    $\delta(t,j) = \max_{i \in Pred(j)} \delta(t-1,i) * a_{i,j} * b_{i,j}(y(t))$ `-- max over every` i `predecessor of` j

For the computation of $\delta(t,j)$, the computation of each $b_{i,j}(y_t)$ takes a constant time[1] $\Theta(1)$ independently of i, j or y_t and thus the cost of this computation is the cost of (1 comparison, 2 multiplications and 1 computation of $b_{i,j}(y_t)$) times the number of predecessors of j, *i.e.* $\Theta(\sharp Pred(j))$.

The cost of the j loop is thus $\Theta(\sum_j \sharp Pred(j))$. If τ denotes the number of transitions in the HMM, then this is equal to $\Theta(\tau)$. Lastly the cost of the t loop is T times this cost, *i.e.* the *a priori* complexity of the

[1] The notation $\Theta(n)$ means that the considered quantity is bounded by $C_1 n$ from below and $C_2 n$ from above, C_1 and C_2 being two constants.

pronunciation is $\Theta(\tau T)$. It is thus proportional to the length of the pronunciation, since τ depends only on the HMM and not on the utterance.

The *a priori* estimated complexity of a word is defined as the sum of the *a priori* complexities of its utterances in the training database.

cTLHMM

The algorithm for the cTLHMM is, for a pronunciation of length T:

```
for t = 1 to T do
    for j = 1 to N do
       (1) δ(t,j,1) = max_{i∈Pred'(j)} max_{1≤d'≤d_max} δ(t-1,i,d') * a_{i,j} * b_{i,j}(y(t)) * ρ_{φ(i)}(d')
                      -- where Pred'(j) is the set of predecessors i of j
                      -- belonging to another phonetic unit φ(i) than j
       (2) for d = 2 to d_max do
          δ(t,j,d) = max_{i∈Pred''(j)} δ(t-1,i,d-1) * a_{i,j} * b_{i,j}(y(t))
                      -- where Pred''(j) is the set of predecessors i of j
                      -- belonging to the same phonetic unit as j
```

The cost of (1) is $d_{\max}\times$ (1 comparison, 3 multiplications, 1 computation of $b_{i,j}(y_t)$, 1 computation of $\rho_{\phi(i)}(d')$) times the cardinal of $Pred'(j)$ *i.e.* $\Theta(d_{\max} \times \sharp Pred'(j))$. The cost of (2) is $d_{\max}\times$ (1 comparison, 2 multiplications, 1 computation of $b_{i,j}(y_t)$) times the cardinal of $Pred''(j)$, *i.e.* $\Theta(d_{\max} \times \sharp Pred''(j))$. The complexity of (1) and (2) is $\Theta(d_{\max} \times \sharp Pred(j))$ where $Pred(j)$ is the set of every predecessor of j, not regarding the phonetic units. There is an extra multiplicative factor $d_{\max}$ for this step compared to the HMM case. The global *a priori* complexity of a pronunciation is thus, with the same notations as before, $\Theta(d_{\max}\tau T)$. It is again proportional to the length of the utterance.

The *a priori* complexity of a word is the sum of the *a priori* complexities of its pronunciations.

4.2 Load-balancing algorithm

Since the *a priori* load estimations of the previous subsection are only approximations, there is no need to implement an optimal distribution strategy and a fast and heuristic one will suffice [5].

The most classical heuristic is a greedy one (called LPT for Largest Processing Time) and the results presented in this paper are obtained with this heuristic. The words are sorted by decreasing complexity. The words are iteratively assigned to the processors according to their order: the first unassigned word is assigned to the processor having the smaller cumulated workload (the tie-breaking rule consists in choosing the processor with the smallest identifier in our implementation), and this is repeated until every word is assigned to a processor. Since the number of words is not very high in a small to medium size vocabulary, the cost of sorting the words is negligible compared to the overall cost of the algorithm. The performances of such an heuristic are well known: the largest workload is less than 4/3 times the largest workload in the optimal distribution [8].

4.3 Experimental results

The experimental times of the parallelizations with static load-balancing are shown on figure 5. When comparing the times with those of figure 3, one notices the gain for the replication strategy, which ranges from 18% (with 4 processors) to 48% (with 10 processors) for HMM, and from 20% (with 2 processors) to 48% (with 7 processors) for cTLHMM. With the distribution strategy the gain of time is less impressive but nonetheless non negligible: between 4.5% and 12% for HMM and between 3% and 12% for cTLHMM. This corroborates our remark in §3.5: the load imbalance was quite high for the replication strategy.

The experimental efficiencies are shown on figure 6. They are much higher than on figure 4, with no efficiency below 65%. The fact that the greedy load balancing improved relatively few the distribution parallelization may be explained by the already good efficiency achieved without load balancing.

The execution times of the fastest processor and the slowest processor are reported in the table below, along with the relative difference ($\frac{t_{slow}-t_{fast}}{t_{slow}}$).

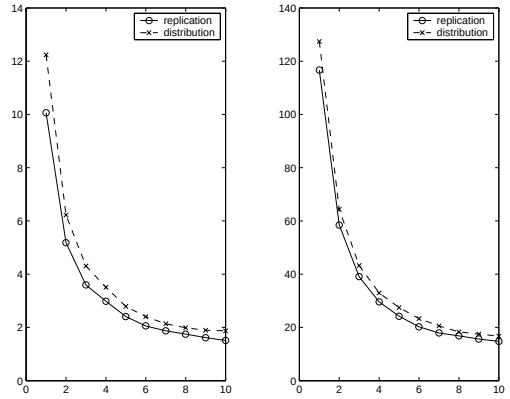


Figure 5: Times in s. versus the number of processors for the replication and the distribution strategies, with HMM (left) and cTLHMM (right).

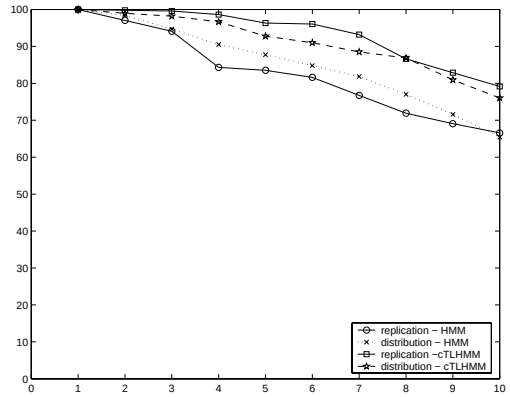


Figure 6: Efficiency for the replication and the distribution strategies using greedy load-balancing, with HMM and cTLHMM.

The most important imbalance was attained with 4 processors (cf. table 1), it has decreased from 56.7% to 7.7% in the replication strategy with HMM and from 40.1% to 6.1% in the distribution strategy with HMM. The corresponding figures for cTLHMM are a decrease from 73.7% to 26.7% with the replication strategy and from 45.0% to 6.8% with the distribution strategy. The estimated imbalance (obtained from the *a priori* load estimations) is in either case 3.8%.

4.4 Comments

A first point which is to be noted is that the communications take a negligible time on the architecture used for the experiments: for the distribution strategy, they take between some percents of the total time on a classical Ethernet network to 0.3% on the fast Myrinet network, cf. tables 3 and 4 which show the overhead due to the parallelization: they give the times of the communications and reestimations of the parameters. Hence, we made no particular effort on this aspect (there is no clever overlapping for instance).

Secondly, the experiments put in evidence that the static load balancing has led to a significant reduction of the imbalance, even if it has not completely suppressed it. A first explanation to this phenomenon is that the number of words is too small to enable a load-balancing without great discrepancies between the heaviest load and the lightest one. Indeed, since the number of words is fixed and quite small (equal to 50 in our experiments), this theoretical imbalance increases with the number of processors and reaches 20% with 20 processors. However, even the optimal load balancing exhibits an imbalance close to this on 20 processors. Since the loads are quite uniform – there is a maximal factor of 3.5 between the highest *a priori* complexity

and the lowest one, with 50 words – with more words it is indeed possible to diminish this discrepancy. It is actually our aim to process a vocabulary of several hundreds of words. Another possible explanation could be the bad quality of the *a priori* complexity estimations. A comparison between theoretical and experimental complexities is made possible by table 5 and figure 7. In this table, the words (which are the fifty first natural numbers, in French) are ordered by increasing estimated complexities; in the right hand-side part of the table, the mean is subtracted from every complexity and the coefficient appearing in the linear fitting formula is used for scaling. It appears that the *a priori* estimations reflect quite well the actual complexities: indeed, correlation coefficients are respectively 0.9801 between the *a priori* estimation loads and the observed complexities for the HMM and 0.9926 between the *a priori* estimation loads and the observed complexities for the cTLHMM. Thus, there is no need to find new estimation formulae that would take into account the expected number of considered transitions and the induced volume of communications for instance, in addition to the computational load.

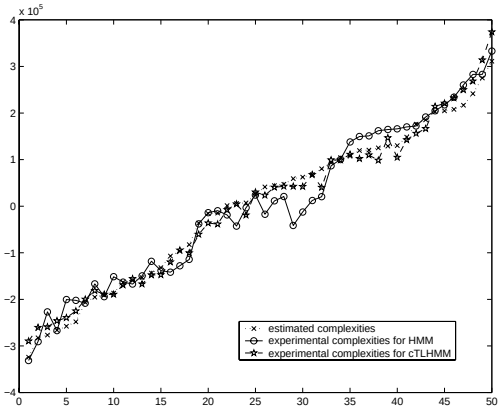


Figure 7: After scaling, the estimated and observed complexities.

The opportunity of dynamic load balancing does not appear very promising, since it would imply a quite expensive mechanism when reliable load estimations are available and can be statically used. Furthermore, it would require communications of parts of the network when the distribution strategy is used, which would take time.

Anyhow, if even the theoretical imbalance should remain too significant, then a further balancing would consist in distributing not only the words among the processors, but also their utterances. This could be seen as a finer grain of parallelism.

Some experiments have been conducted on the parallelization of the recognition stage. The figure 8 shows the efficiencies of the recognition of the word "45", which belongs to the vocabulary, using an utterance not in the database. This word "quarante-cinq" has an intermediate length. The efficiencies are rapidly decreasing when the number of processors increases, which may be due to the very small size of the local networks and thus to the very short computational time compared to the communication time.

5 Conclusion and future work

The problem of accurate recognition of small- to medium-size vocabularies can be tackled by the use of models that take into account the prosody of speech, such as the cTLHMM model briefly recalled here. The need for parallelizing the task of automatically recognizing the speech has been motivated and then two strategies of parallelization for automatic speech recognition have been proposed and experimented. The first one can be applied to the training stage only, the second one can be applied to the training and recognition stages. Preliminary experiments have put in evidence the need for a load-balancing that take into account the computational work implied by each word and not only the number of words assigned to each processor. Our *a priori* load estimation corresponds to the length of the word: this intuitive estimation has been confirmed by the analysis of the algorithm. A fast and heuristic static load-balancing algorithm,

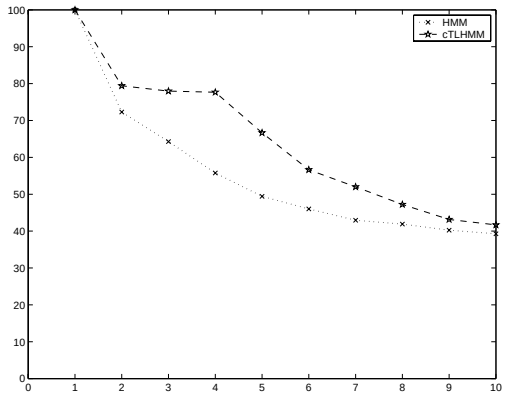


Figure 8: Efficiency for the recognition of the word "45", with HMM and cTLHMM.

based on these *a priori* load estimations, has led to significant improvements of the parallel performances.

Another parallelization strategy that would combine the advantages of the two strategies presented in this paper is the following one:

- the network is distributed among the processors as in the distribution strategy, in order to use efficiently all the memory available;
- the probabilities of the phonetic units are shared, *i.e.* a given pseudo-diphone has the same laws whatever the word or the processor where it appears: this avoids the drawbacks of the distributed strategy, namely the need for a larger training database since there are more parameters to estimate and also the longer processing time for one iteration which has been experimentally observed. It implies a larger volume of communications, but it has also been experimentally put in evidence that the communications take a negligible time.

This strategy constitutes our next implementation and experimentation step.

Another parallelization strategy would be to adopt a completely different point of view and to parallelize Viterbi algorithm. It would consist in parallelizing the processing of each word and in serializing the words. This implies the distribution of the network, each processor being in charge of a subset of states. The expected problems are on the one hand the volume of communications required to compute, at each step t, the maximal value for $\delta(t, j)$ (cf. §3.2) and on the other hand the poor scalability of this parallel algorithm; indeed there must be at least as many states j to examine in parallel as there are processors and this assumption cannot hold when the number of processors increases.

References

[1] F. Chaussumier, F. Desprez, and L. Prylli. Asynchronous communications in MPI - the BIP/Myrinet approach. In LNCS, editor, *Proc. 6th European PVM/MPI Users' Group*, volume 1697, pages 485–492, 1999.

[2] M. Cravero, R. Pieraccini, and F. Raineri. Definition and evaluation of phonetic units for speech recognition by hidden Markov model. In *Proc. IEEE International Conference ASSP'86, Tokyo*, 1986.

[3] T.H. Crystal and A.S. Housse. Segmental durations in connected speech signals: preliminary results. *J. Acoustic Soc. Am.*, 72(3):705–716, 1982.

[4] T.H. Crystal and A.S. Housse. Segmental durations in connected speech signals: current results. *J. Acoustic Soc. Am.*, 83(4):1553–1573, 1988.

[5] E.M. Daoudi, P. Manneback, A. Meziane, and Y.O. Mohamed El Hadj. Study of the load-balancing in the parallel training for automatic speech recognition. In LNCS, editor, *EuroPar'2000*, volume 1900, pages 506–510, 2000.

[6] E.M. Daoudi, A. Meziane, and Y.O. Mohamed El Hadj. Study of parallelization of the training for automatic speech recognition. In LNCS, editor, *HPCN 2000*, volume 1823, pages 576–579, 2000.

[7] M. Fleury, A.C. Downton, and A.F. Clark. Parallel structure in an integrated speech-recognition network. In LNCS, editor, *EuroPar'99*, volume 1685, pages 995–1004, 1999.

[8] R.L. Graham. Bounds for multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.

[9] S.E. Levinson. Continuously variable duration hidden Markov models for automatic speech recognition. *Computer Speech and Language*, 1:29–45, 1986.

[10] A. Meziane. *Introduction de la durée des sons dans un modèle de Markov caché au niveau supra-segmental*. Doctorat d'état es sciences, Univ. Mohamed 1st, Faculty of sciences Oujda, 1997.

[11] A. Meziane, B. Jacob, and R. André-Obrecht. Modélisation de la durée des sons dans un système de reconnaissance automatique de la parole. *C.R.A.S.*, 327(IIb):379–382, 1999.

[12] J.L. Miller. *Effects of speaking rate on segmental distinctions*, chapter Perspective of the study of speech, pages 39–74. Lawrence Erlbaume Associates, P.D. Eimas and J.L. Miller (eds) edition, 1981.

[13] C. Mokbel, L. Mauuary, L. Karray, D. Jouvet, J. Monné, J. Simonin, and K. Bartkova. Towards improving ASR robustness for PSN and GSM telephone applications. *Speech communication*, 23:141–159, 1997.

[14] H. Noda, M.N. Shirazi, and B. Zhang. A parallel processing algorithm for speech recognition using Markov random fields. *Journal of the communications research laboratory*, 41(2):87–100, July 1994.

[15] D. O'Shaughnessy. A study of french vowel and consonant durations. *Journal of Phonetics*, 9:385–406, 1981.

[16] S. Phillips and A. Rogers. Parallel speech recognition. In *EuroSpeech'97*, volume 1, pages 135–138, 1997.

[17] L.R. Rabiner. An introduction to Hidden Markov Models. *IEEE Trans. ASSP*, 77(2):257–286, 1989.

[18] M.J. Russel and A.E. Cook. Experimental evaluation of duration modelling techniques for automatic speech recognition. In *Proc. IEEE ICASSP*, pages 2376–2379, 1987.

[19] N. Suaudeau and R. André-Obrecht. An efficient combination of acoustic and supra-segmental information in a speech-recognition system. In *ICASSP'94*, pages 65–68, April 1994.

[20] A. Tuerk and S. Young. Modelling speaking rate using a between frame distance metric. In *Eurospeech'99*, volume 1, 1999.

[21] J. Vaissiere. *Computer speech processing*, chapter Speech recognition: a tutorial, pages 191–236. Prentice Hall, f. fallside and n.a. words (eds) edition, 1986.

[22] J. Vaissiere. *Recend advances in speech understanding and dialog systems*, volume F46 of *DF*, chapter The use of prosodic parameters in automatic speech recognition, pages 71–100. NATO ASI Series, H. Nieman et al. edition, 1988.

[23] J.P.H. Van Santen and J.P. Olive. The analysis of contextual effects on segmental duration. *Computer Speech and Language*, 4:359–390, 1990.

[24] S. Young. A review of large vocabulary continuous-speech recognition. *IEEE Signal Processing magazine*, 13(5):45–57, 1996. Also available on `http://svr-www.eng.cam.ac.uk/reports/index-speech.html`.

Nb of proc.	Replication strategy			Distribution strategy		
	time of the fastest in s.	time of the slowest in s.	relative diff. in %	time of the fastest in s.	time of the slowest in s.	relative diff. in %
HMM						
2	3.682	6.561	43.881	4.245	6.552	35.211
3	2.046	4.532	54.854	2.884	4.757	39.374
4	1.403	3.239	56.684	2.243	3.742	40.059
cTLHMM						
2	39.003	73.004	46.574	41.259	66.476	37.934
3	23.483	50.802	53.775	25.758	45.886	43.865
4	10.384	39.465	73.688	20.146	36.628	44.998

Table 1: Measure of the load imbalance between the fastest and the slowest processors with the replicated and distributed strategies.

Nb of proc.	Replication strategy			Distribution strategy		
	time of the fastest in s.	time of the slowest in s.	relative diff. in %	time of the fastest in s.	time of the slowest in s.	relative diff. in %
HMM						
2	4.850	5.131	5.477	6.145	6.210	1.047
3	3.431	3.487	1.606	4.215	4.303	2.045
4	2.470	2.675	7.664	3.285	3.500	6.143
cTLHMM						
2	54.080	58.428	7.442	63.410	64.353	1.465
3	34.245	38.876	11.912	42.035	43.233	2.771
4	21.385	29.189	26.736	30.706	32.950	6.810

Table 2: Measure of the quality of the greedy load-balancing between the fastest and the slowest processors with the replicated and distributed strategies.

Nb of proc.	Replication strategy			Distribution strategy		
	total exec. time in s.	comm. + reestim. time in s.	relative importance of comm. + reestim. in %	total exec. time in s.	comm. + reestim. time in s.	relative importance of comm. + reestim. in %
HMM						
2	6.597	0.035	0.531	6.588	0.016	0.243
3	4.701	0.170	3.616	4.756	0.009	0.189
4	3.639	0.403	11.074	3.749	0.007	0.187
cTLHMM						
2	73.034	0.072	0.099	66.492	0.023	0.035
3	50.846	0.193	0.380	45.896	0.015	0.033
4	39.772	0.452	1.136	36.636	0.011	0.030

Table 3: Measure of the overhead due to the parallelization: relative importance of the communications and reestimations of the parameters compared to the total execution time with the replicated and distributed strategies, without load-balancing.

Nb of proc.	Replication strategy			Distribution strategy		
	total exec. time in s.	comm. + reestim. time in s.	relative importance of comm. + reestim. in %	total exec. time in s.	comm. + reestim. time in s.	relative importance of comm. + reestim. in %
HMM						
2	5.184	0.018	0.347	6.223	0.013	0.209
3	3.594	0.109	3.033	4.310	0.008	0.186
4	2.983	0.314	10.526	3.506	0.006	0.171
cTLHMM						
2	58.481	0.058	0.099	64.376	0.020	0.031
3	39.060	0.145	0.371	43.253	0.012	0.028
4	29.585	0.399	1.349	32.950	0.009	0.027

Table 4: Measure of the overhead due to the parallelization: relative importance of the communications and reestimations of the parameters compared to the total execution time with the replicated and distributed strategies, with load-balancing.

Word	Complexities			Centered and scaled complexities		
	estimated	observed for HMM	observed for cTLHMM	estimated	observed for HMM	observed for cTLHMM
1	2.4824	0.1159	0.8440	-3.2423	-3.3124	-2.8968
3	2.8980	0.1437	0.9818	-2.8266	-2.9054	-2.6037
2	2.9566	0.1873	0.9875	-2.7681	-2.2679	-2.5915
9	3.0622	0.1597	1.0489	-2.6624	-2.6717	-2.4610
6	3.1472	0.2052	1.0797	-2.5774	-2.0053	-2.3954
20	3.2435	0.2041	1.1474	-2.4811	-2.0226	-2.2514
7	3.6600	0.1999	1.2664	-2.0646	-2.0840	-1.9983
8	3.7715	0.2283	1.3530	-1.9531	-1.6681	-1.8141
10	3.8186	0.2095	1.3159	-1.9060	-1.9429	-1.8930
12	3.8571	0.2388	1.3159	-1.8675	-1.5154	-1.8931
15	4.0880	0.2311	1.4100	-1.6366	-1.6274	-1.6929
4	4.1278	0.2281	1.4731	-1.5969	-1.6710	-1.5587
13	4.1925	0.2401	1.4197	-1.5321	-1.4960	-1.6724
30	4.2892	0.2616	1.5120	-1.4354	-1.1813	-1.4760
16	4.4042	0.2479	1.5136	-1.3204	-1.3813	-1.4726
11	4.6543	0.2455	1.6423	-1.0704	-1.4170	-1.1990
5	4.7583	0.2550	1.7612	-0.9663	-1.2785	-0.9460
0	4.8986	0.2642	1.7345	-0.8261	-1.1433	-1.0029
32	5.3618	0.3165	1.9216	-0.3629	-0.3787	-0.6049
33	5.5671	0.3329	2.0358	-0.1575	-0.1388	-0.3620
40	5.5757	0.3358	2.0253	-0.1490	-0.0958	-0.3845
38	5.7375	0.3296	2.1712	0.0129	-0.1875	-0.0742
18	5.7821	0.3132	2.2280	0.0575	-0.4262	0.0467
19	5.7957	0.3401	2.1157	0.0711	-0.0329	-0.1922
28	5.9792	0.3587	2.3482	0.2546	0.2385	0.3022
31	6.1382	0.3306	2.3155	0.4135	-0.1731	0.2328
23	6.1667	0.3500	2.3972	0.4421	0.1119	0.4066
22	6.1948	0.3566	2.4064	0.4701	0.2074	0.4260
36	6.3150	0.3141	2.4057	0.5904	-0.4138	0.4245
14	6.3485	0.3340	2.4039	0.6239	-0.1234	0.4207
29	6.4032	0.3508	2.5243	0.6786	0.1230	0.6767
17	6.5283	0.3563	2.3954	0.8037	0.2040	0.4027
27	6.6588	0.4017	2.6724	0.9342	0.8679	0.9919
26	6.7686	0.4111	2.6728	1.0440	1.0042	0.9926
25	6.7977	0.4368	2.7289	1.0731	1.3803	1.1120
37	6.9187	0.4446	2.6861	1.1940	1.4948	1.0210
21	6.9287	0.4456	2.7224	1.2041	1.5090	1.0982
39	6.9754	0.4532	2.6692	1.2508	1.6206	0.9850
34	7.0134	0.4549	2.8957	1.2888	1.6446	1.4666
42	7.0253	0.4560	2.6986	1.3007	1.6606	1.0474
35	7.2075	0.4588	2.8757	1.4829	1.7024	1.4241
43	7.4849	0.4601	2.9403	1.7603	1.7207	1.5616
48	7.5776	0.4733	2.9883	1.8530	1.9139	1.6636
41	7.7512	0.4825	3.2125	2.0266	2.0479	2.1405
44	7.7710	0.4912	3.2449	2.0464	2.1753	2.2094
24	7.8033	0.5025	3.2992	2.0787	2.3404	2.3247
49	7.8924	0.5206	3.3810	2.1677	2.6051	2.4987
47	8.1396	0.5359	3.4682	2.4150	2.8290	2.6841
46	8.4771	0.5359	3.6813	2.7525	2.8288	3.1374
45	8.8358	0.5702	3.9641	3.1112	3.3314	3.7389

Table 5: Estimated and observed complexities (on the right hand-side, their mean is subtracted and the linear fitting coefficient is applied).