POST : Parallel Object-Oriented Speech Toolkit

Jean Hennebert and Dijana Petrovska Delacrétaz
Circuits and Systems Group - Swiss Federal Institue of Technology, Lausanne

ABSTRACT

We give a short overview of *POST*, a parallel speech toolkit that is distributed freeware to academic institutions. The underlying idea of *POST* is that large computational problems, like the ones involved in Automatic Speech Recognition (ASR), can be solved more cost effectively by using the aggregate power and memory of many computers.

In its current version (January 96) and amongst other things, *POST* can perform simple feature extraction, training and testing of word and subword Hidden Markov Models (HMMs) with discrete and multigaussian statistical modelling.

In this parer, the implementation of the parallelism is discussed and an evaluation of the performances on a telephone database is presented. A short introduction to Parallel Virtual Machine (PVM), the library through which the parallelism is achieved, is also given.

1. Introduction

POST is a Parallel Object-oriented Speech Toolkit developed for designing speech and speaker recognition applications. The idea of the *POST* toolkit is born starting from the following observations. First, there is a growing demand for computer resources in order to build more complicated speech models and to deal with very large databases. Second, ASR applications can easily be parallelized at several levels of the algorithms. Finally, very efficient tools to parallelize networked Unix computers are available.

The *POST* project has started in September 95 within the Circuits and Systems Group of the Swiss Federal Institute of Technology, Lausanne. This toolkit, primarily designed for internal purposes, presents new features (mainly parallelism) that other speech toolkits does not have. Therefore, it has been decided to provide freeware executable of it to academic institutions[1].

The main objective of *POST* is to deal with heavy speech/speaker recognition tasks. More precisely, *POST* is a research and development speech toolkit built in order to perform reliable comparisons of different algorithms and to develop, compute and test quickly new speech models. From a hardware point of view, *POST* is a low-cost speech toolkit that works in parallel on an heterogeneous collection of Unix computers without any dedicated hardware.

This paper is organized as follows. In the second section, PVM, the library through which the parallelism is achieved, is presented. In the third section, the choice of the parallel architecture is discussed. The fourth section gives a closer look of the features already implemented into *POST*. In the fifth section, we present results of a speaker independent isolated word recognition task[1].

2. Parallel Virtual Machine

POST achieves parallelism through a freeware library called Parallel Virtual Machine (PVM) [7]. We have choosen PVM because it is a wide-spread, well validated, simple and flexible tool. The PVM distribution including sources and documentation can be downloaded from [2][2].

In short, PVM is a software system that enables a collection of heterogeneous computers to be used as a coherent and flexible concurrent computational resource. Such a parallel system is referred to a *Multiple Instruction stream, Multiple Data stream* (MIMD) system in which there are N tasks, N streams of instructions and N streams of data [4]. Each processor is potentially executing *asynchronously* different programs on different data while solving different subtasks of a single task. PVM can deal with shared-memory *multiprocessors systems* (or *tightly coupled machines*) or local-memory processors interconnected by a network (*multicomputers* or *loosely coupled machines*). A set of functions such as process initiation, message transmission and reception, and process synchronization is provided. The PVM system transparently handles message routing, data conversion for incompatible architectures, and other tasks that are necessary for opera-

[1]This task can not be considered as a large ASR task but we have chosen it in order to have a simple and fast validation of the algorithms installed in the toolkit.

[2]Other similar packages do exist, as the Message Passing Interface [3], that seem to be very efficient to implement parallelism.

tion in a heterogeneous, network environment.

The PVM software is very portable and has been compiled on everything from laptops to CRAYs. PVM enables users to exploit their existing computer hardware to solve much larger problems at minimal additional cost. Hundreds of sites around the world are using PVM to solve important scientific, industrial, and medical problems. The PVM system has been used for applications such as molecular dynamics simulations, superconductivity studies, distributed fractal computations, matrix algorithms, and in the classroom as the basis for teaching concurrent computing.

3. Parallelism within *POST*

Generally, the choice of a parallel computer processing configuration is dependent of what we want to solve and of the hardware we have at disposition[4]. Looking at common ASR experiments, the first kind of task that comes to our mind is going through a large database in order to produce an evaluation of a feature extraction algorithm, a statistical model or a grammar network. We call these kind of tasks *database tasks*. The second task is doing online experimentations in order to evaluate the user side of ASR applications. We call these kind of tasks *online tasks*.

From the hardware point of view, our purpose is to do an optimal usage of the cpu power of a cluster of Unix workstations, taking into account the following observations :

1. they may come from different familly or be from different generation of processors;
2. they may be of different cpu capabilities (other processes running or different kind of processors);
3. they are linked in some manner (local area network, multiprocessor system, ...).

Considering the hardware constraints listed above, it is clear that the purpose of *POST* is not to be a real-time application software. Nevertheless, it should permit very fast online demonstrations by using the aggregate power and memory of many computers.

Database tasks require an optimization of the cpu resources while for online tasks, optimization of the response time is necessary. Both tasks will need to be treated differently by the parallel system. Given the application and hardware constraints described above, we have decided for a master-slave configuration as illustrated in figure 1. The master is running on the processor where the command has been issued and N slaves are launched on P processors.

As described hereafter, this configuration enables the treatment of both database tasks and online tasks. In its current version, *POST* can perform database tasks in parallel. The implementation of online tasks is still under development.

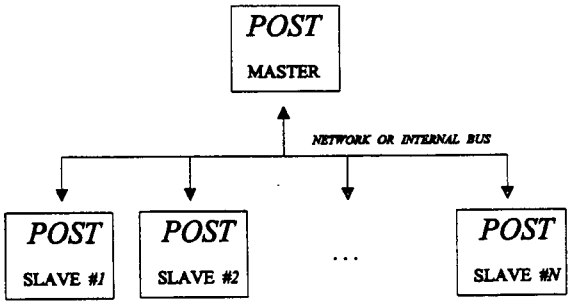

Figure 1: Master-slave configuration of POST with N slaves.

3.1. Database tasks

The global process or task is split into a set of T independent subtasks, so that $T \gg N$. By independent subtask, we mean a subtask that does not need to wait the result of another subtask to be processed. The operations are the following:

1. Initialisation: the master sends to each slave the necessary information about the subtask he has to process;
2. The slaves execute in parallel the code in order to solve their subtask;
3. As soon as a slave has finished his job, he warns the master;
4. The master accumulates the subtask results and gives to the freed slave another subtask to process;
5. Go back to 2.

If some slaves are more powerful (better processor or processor less loaded), they will just perform more subtasks. This way of distributing subtasks leads to a natural automatic load balancing and to an optimal usage of the global cpu resources.

3.2. Online tasks

In this case, the task is split into a set of T dependent subtasks, so that $T \simeq N$. By dependent subtask, we mean a subtask that needs to wait for the result of another subtask to be processed. This configuration is one of a pipeline in which the master orchestrates the data flow between the slaves. The operations are:

1. Initialisation: the master sends to the each slave the necessary information about the subtask he has to process;
2. The slaves execute in parallel the code in order to solve the subtask;
3. As soon as a slave n has finished his job, he warns the master;

	Task	Mode
Parametrization	D	S/P
Quantization	D	S/P
HMM Training	D	S/P
Isolated Word Reco	D	S/P
Isolated Word Reco	O	S
Connected Word Reco	O	planned
HMM Annotation	D	S

Table 1: Features implemented into POST.

4. The master passes the results of the subtask issued from slave n to the slave $n+1$ in the pipe;
5. Go back to 2.

This way of distributing subtasks does not generally lead to an optimal usage of the global cpu resource because some slaves may wait uselessly if the preceding one in the pipe has not finished with his subtask (bottleneck problem). Automatic techniques for resizing subtasks and for spawning new slaves where needed are under investigation.

4. Features of *POST*

The toolkit is developed in C++ and should be portable on the majority of UNIX workstations without any problems. The object oriented programming concept has been chosen to facilitate multi-user developments and future extensions of the toolkit.

In its current version (January 96), *POST* can solve database tasks (D) and online tasks (O), in standalone mode (S) and/or parallel mode (P). Table 1 gives an overview of the kind of ASR tasks that *POST* can handle.

The algorithms that are currently implemented are the following:

- Liftered LPC-Cepstral feature extraction [11];
- Viterbi HMM training, Viterbi HMM recognition [12];
- Discrete[3] [8], Multi-Gaussian [10] and Parzen Windows [5] probability density function models;
- Context Independent Phoneme (CIP) models, Context Dependent Phoneme (CDP) models or Word models [12].

We distinguish between the task and its algorithm solving because the object oriented way of programming permits to exchange very easily the algorithm engine. For a more complete description of the algorithms and of POST in general, the reader should refer to [9].

[3]Codebook creation is currently performed outside of the toolkit.

5. Results

We report here a preliminary evaluation of the performance of the toolkit with a speaker independent isolated word recognition task. For an extended version of these results, the reader can take a look in [1]. All the experiments reported in this paper were performed using the same training and test set in order to ensure reliable comparisons.

We used a database in German language containing 108 phonetically balanced words pronounced by 536 speakers. 400 speakers were used to train the HMMs, 136 speakers were used as test set. We used the liftered LPC-cepstral feature extraction algorithm with cepstral mean substraction (LCMS). The acoustic vectors were computed every 10 ms over a 30-ms window. After pre-emphasis (α=1) and application of a Hamming window, ten LPC coefficients were used to compute the cepstrum.

The first set of results reported in table 2 shows the influence of the vocabulary size. Tests conditions are the following: 4 streams (12 LCMS, 12 ΔLCMS, 1 Δlog(E), 1 $\Delta\Delta$log(E)), monogaussian statistical modelling, 42 CIP models with 3 states/CIP plus one state model for the silence, CIP models training performed on 108 words with 15 iterations on the database, testing performed on words models obtained from a concatenation of CIP models. The quite low performance on the digits is explained by the high confusion that exists between the german words "zwei" and "drei". We have to point out that, for this kind of isolated word recognition task, the response time can be independent of the vocabulary size if enough computers are used. Indeed, the recognition is performed computing the best score through all of the models which can be efficiently spread by the toolkit through the cluster of workstations.

Vocabulary Size	Error Rate
10 (digits)	6.6 %
108	9.3 %

Table 2: Comparison results regarding the vocabulary size of an isolated word recognition.

Table 3 shows the gain in performance with the addition of delta and energy coefficients to the single cepstrum vector stream. Tests conditions are the following: discrete statistical modelling, 10 words models (digits), codebooks trained with the K-means algorithm initialized by a Kohonen algorithm [6], codebook sizes are 121 for LCMS, 121 for ΔLCMS, 32 for $\Delta\log(E)$ and 32 for $\Delta\Delta\log(E)$.

Table 4 shows the performances of different statistical models. Tests conditions are the following: 1 stream of 12 LCMS coefficients, 10 word models (digits). As expected, multigaussian modelling performs better than monogaussian and discrete systems.

Table 5 shows the performance of different (sub)word models.

Data Streams	#Stream	Error Rate
12 LCMS	1	8.1 %
12 LCMS $+\Delta\log(E)$	2	3.7 %
12 LCMS+12 ΔLCMS $+\Delta\log(E) + \Delta\Delta\log(E)$	4	2.6 %

Table 3: Comparison results regarding the number of streams.

State Statistical Model	Error Rate
Discrete	8.1 %
Monogaussian	5.4 %
4 Mixtures	4.8 %

Table 4: Comparison results regarding the state statistical model.

Tests conditions are the following: 4 streams (12 LCMS, 12 ΔLCMS, 1 Δlog(E), 1 $\Delta\Delta$log(E)), discrete statistical modelling, 108 words for training and testing, codebooks trained with the K-means algorithm initialized by a Kohonen algorithm, codebook sizes are 121 for LCMS, 121 for ΔLCMS, 32 for $\Delta\log(E)$ and 32 for $\Delta\Delta\log(E)$, CDP transcriptions obtained by a simple concatenation of two adjacent CIPs. As expected, word models perform better than CDP models that perform better than CIP models. On the other hand, CIP and CDP models permit to extend the vocabulary. There is a tradeoff between the performance, the flexibility and the number of states the system has to handle.

Model Type	#Models	#States	Error Rate
Word	108	2133	2.7 %
CDP	349	1047	5.0 %
CIP	42	126	7.1 %

Table 5: Comparison results regarding the model type.

6. Conclusion and Future Work

In this paper, we presented an overview of *POST*, a parallel object-oriented speech toolkit. We gave a short description of the kind of task we would like to handle in parallel with the toolkit, namely database tasks and online tasks. We discussed the master-slave architecture choice and we reported results on a speaker independent task that seem to validate the algorithms installed in the toolkit.

In its current version, the toolkit can do usual database tasks in parallel mode and online tasks in standalone mode. While *POST* is still at an early stage of its development, freeware executable of *POST* are made available for academic institutions[1].

In the future extensions of the toolkit, the following features are planned to be released:

- input-output compatibility with standart databases,
- connected word recognition,
- hybrid HMMs-neural networks,
- online parallel tasks with automatic bottleneck cancellation technique.

7. Acknowledgments

We would like to thank F. Bugnon, J. Reichel, P. Pfister and A. Sanchez for their helpful contribution in testing and debugging the toolkit. We acknowledge the CIRC and EPFL in general for supporting this work.

8. REFERENCES

1. http://circwww.epfl.ch/data/general/jean/post/post.html.
2. http://www.epm.ornl.gov/pvm/.
3. http://www.erc.msstate.edu/mpi/.
4. S. G. Akl. *The Design and Analysis of Parallel Algorithms.* Prentice-Hall, 1989.
5. Duda and Hart. *Pattern Classification and Scene Analysis.* Wiley and Sons, 1973.
6. V. Fontaine, J. Hennebert, and H. Leich. Influence of vector quantization on isolated word recognition. In *Proceedings of Eusipco*, pages 115–118, Edinburgh, 1994.
7. A. Geist and al. *PVM: Parallel Virtual Machine, A Users'Guide and Tutorial for Networked Parallel Computing.* MIT Press, 1994.
8. A. Gersho and R. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, 1992.
9. J. Hennebert. *Manual of the Parallel Oriented Speech Toolkit.* Chaire des Circuits et Systemes, 1996.
10. X. D. Huang, Y. Ariki, and M. A. Jack. *Hidden Markov Models for Speech Recognition.* Edinburgh University Press, 1990.
11. J. Picone. Signal modeling techniques in speech recognition. *Proceedings of the IEEE*, 81(9):1214–1247, September 1993.
12. L. Rabiner and B.-H. Juang. *Fundamentals of Speech Recognition.* Prentice Hall, 1993.