Multiplatform Server-Based Speech Recogniser Generator for Embedded Systems

Robert Krejčí¹, Václav Hanžl²

¹Department of Circuit Theory, Electrical Engineering Faculty, Czech Technical University, Technická 2, 166 27 Praha, Czech Republic robert.krejci@centrum.cz ²Department of Circuit Theory, Electrical Engineering Faculty, Czech Technical University, Technická 2, 166 27 Praha, Czech Republic

hanzl@noel.fel.cvut.cz

Abstract – Automatic speech recognisers still more come in the practice of regular users of computers and various devices. However, so far there is no speech recogniser on the market in the form of an electronic component or a module that could be easily usable in embedded systems. Our intention is to prepare a theoretical concept and practical realisation of a software tool that facilitates the development of the standalone module of spoken commands recogniser and enables the easy configurability during a possible purchase in an eshop for each individual part. The first step towards this goal is our phpHMM software tool that facilitates the development and optimisation of speech recogniser algorithms. We use it for generating source code of speech recogniser using the PHP scripting language and MvSOL database. The input of the system is a speech model and a list of words to be recognised. The output is the source code and data structures in the C programming language, which are consecutively compiled into an executable program.

I. INTRODUCTION

In modern devices increasingly emerges a new possibility of controlling by voice commands. This feature can lead to miniaturisation of these devices, elimination of mechanical components, it improves comfort of handling and it can have a positive effect on reliability and price. So far, however, there is no general electronic component or module for speech recognition on the market in the sense of commonly used Bluetooth, WiFi or GSM modules. The main reason is that speech recognition is computationally very intensive process, especially if it were an entirely general speech without defining the field, and currently there is still not available any computer technology that would simultaneously meet the task requirements of embedded systems – high computing performance, low power, small size, low price.

However, if we put ourselves a task to create the speech recogniser with a limited number of recognised words, as commands for a specific controlled device, we can get closer to the idea of a miniature speech recognition module, which has human speech (command) as input and a text transcription of this speech as output or possibly even value of the logic levels on I/O ports of the module.

To have a good chance of success, it must be easily configurable even for each individually produced module. Our idea of configurability is based on the concept of internet e-shop, where the customer chooses ordered module parameters, including in particular the list of words that the module will be able to recognise. Everything other would then be done automatically: a speech model and choice of hardware platform, assembling the source code and data structures, compilation and generation of executable file.

We are currently developing a basis for such entirely automated system. It is the class of functions, which we called *phpHMM*. Due to the possible future implementation to an internet e-shop, we have chosen the PHP programming language using a MySQL database as the base platform of the tool.

Since at the creation of speech recogniser it is worked with huge amounts of data, many activities are performed automatically using scripts to facilitate the work and eliminate the need for repeated manual data processing. For this serves e.g. a tool known as HTK Toolkit [1], using which it can be created a complete speech recogniser for the PC platform.

However, when creating a speech recogniser, which is run on different hardware platforms, as e.g. digital signal processors, there is no such public tool available and we have to create own one. In this case the speech models trained using the HTK toolkit can be used, but for their treatment totally different algorithms and optimisation methods must be used than those used on the PC platform. To test the optimisation methods there is often necessary to change the data structures and convert their parameters. Therefore, in the Speech Processing Group at the Department of Circuit Theory, CTU in Prague, a "phpHMM" tool, that facilitates and integrates the development of speech recognition algorithms to alternative hardware platforms, is developed.

II. PhpHMM TOOL

PhpHMM tool is a set of scripts in PHP [2] using MySQL database [3]. This technology has become

one of the standards to generate web pages, but of course it can be used for generating also any other texts, such as source code. The base of the *phpHMM* tool is a class of functions that can be easily included into a superior system. The scripts are run on the server (either on a local computer configured as a server or on a publicly accessible web server) and their output is visible via a graphical user-friendly web interface. By a sequence of single steps, source code of speech recogniser can be made. They will be discussed in the following text.

Speech Model

The result of training-phase of the speech recogniser using the HTK is a file in a defined format that describes a general model of speech, created based on the utterances of training database. The models of speech may have a huge number of different variations, such as type of parametrisation, the number of states, streams, and mixtures, the number of coefficients in each mixture, etc. During recognition, these parameters enter the output probability density function b(o) [4]:

$$b_{j}(\overline{o}_{t}) = \prod_{s=1}^{S} \left[\sum_{m=1}^{M_{s}} c_{jsm} N(\overline{o}_{st}; \overline{\mu}_{jsm}, \Sigma_{jsm}) \right]^{r_{s}};$$

$$N(\overline{o}_{st}; \overline{\mu}_{jsm}, \Sigma_{jsm}) = \frac{1}{\sqrt{(2\pi)^{n_{s}} |\Sigma|}} e^{-\frac{1}{2} (\overline{o}_{st} - \overline{\mu}_{jsm})^{T} \Sigma_{jsm}^{-1} (\overline{o}_{st} - \overline{\mu}_{jsm})},$$
(1)

where *S* is count of streams, γ_s is stream weight, M_s is count of mixtures in a stream, c_{jsm} is weight of *m*-th mixture, $N(\overline{o}; \overline{\mu}, \Sigma)$ is multivariate Gaussian distribution with a vector of mean values $\overline{\mu}$ and a covariance matrix Σ .

All these factors enter into the *phpHMM* tool by uploading the file.



Fig. 1: Example of simple hidden Markov model of "a" phoneme in text form

Parsing and Storing into Database

After uploading a text file with hidden Markov models, their parsing follows and conversion from the text form into data structures in the memory of the server. At the same time, some basic integrity checks of the file are carried out. Then database tables are created in the MySQL database and they are populated with relevant data. Using the database is convenient, inter alia, for easy selection of data by means of (even complicated) SQL queries. Selection and processing of data using a database is significantly faster, comfort and more reliable than searching in a text file. For our current experiments it is advantageous to store data in a MEMORY table type, as this storage allows faster access than the commonly used MyISAM type.

Glossary of words

In this step, it can be simply specified all the words which the recogniser will be able to recognise, either by typing in the text-box, or by uploading a text file. The more words are to be recognised, the greater will be the demands on the recogniser hardware, and hence the optimisation of the algorithms.

Phonetic Transcription

In many languages, including Czech, there is the difference between written and spoken form of speech. This step automatically creates a phonetic transcription of words entered in the previous step. Eg. the Czech word "zpěv" will be rewritten by the transcription "spjef".

Selection of Hardware Platform

The intention of the *phpHMM* tool is to create a general tool for a large number of hardware platforms. Currently, it is possible to choose between these platforms:

Table 1: Currently available hardware platforms in phpHMM

Processor	Architecture	Manufacturer
general	32-bit	—
OMAP-L137 (DSP + ARM) 300 + 300 MHz	Dual-core: 32-bit DSP TMS320C674x + 32-bit ARM 9	Texas Instruments
TMS320C5505 100 MHz	16-bit DSP TMS320C55x	Texas Instruments
Stellaris LM3S8962 80 MHz	32-bit ARM Cortex M3	Texas Instruments
STM32F103 72 MHz	32-bit ARM Cortex M3	STMicroelectronics

Particular hardware platforms are briefly described in the following.

➢ OMAP-L137 is a 32-bit dual-core heterogeneous processor, which consists of the ARM 9 core that enables to run an operating system (eg. Linux), and TMS320C674x digital signal processor core with VLIW architecture and with both fixed and floating point hardware arithmetic. [5]

> TMS320C5505 is a 16-bit digital signal processor with very low power consumption suitable for processing audio signals. It also includes an FFT coprocessor. [6]

➤ LM3S8962 and STM32F103 are 32-bit microcontrollers with ARM Cortex M3 core, which also contains some basic DSP instructions (MAC, bit shifts, saturation, bit reverse). [7]

Selection of Optimisation Methods

If we want to run speech recogniser on a system with limited hardware resources, it is necessary to perform optimisation of computationally intensive algorithms. In this step, a combination of optimisation methods can be chosen that we want to be tested. The optimisation is done at all levels of the design of the speech recogniser – from the layout of the data structures up to modifying the algorithms so that they are faster performed on the chosen hardware platform.

Creating Word Models

Depending on the optimisation method, models of the words are created as sequences of states with which the Viterbi algorithm works. For each word, the phoneme models are chained into one sequence of states.

Assembling Source Code and Data Structures

The main task of the *phpHMM* tool is to set up the source code and data structures based on the input data, specification of which has just been described. Depending on the type of parametrisation, structure of the models and the required optimisations, the system generates the sources of the speech recogniser with the relevant data.

The generating of the source code must be before programmed for each selection of hardware platform and optimisation. Setting up of source code can be done using PHP very effectively. The code of the PHP scripting language can be inserted directly into the source code in C. PHP as a pre-processor can be used [8], which has, compared to the standard C preprocessor, much more possibilities – such as creating cycles or computing with geometric functions. For example, Hamming window can be generated as follows:

```
<?php $N=512; ?>
const float hamming_ar[<?php echo $N;?>]={
    <?php // generating lookup table
    for ($n=0; $n<$N; $n++) {
        $w=0.54 - 0.46 * cos(2*pi()*$n/$N);
        echo "$w,";
    }
    ?>
};
```

The generated codes are subsequently compiled by the appropriate compiler. But it is already beyond the function of this tool, although in future it may be possible, after generating the source codes, just run the compiler and get the program in an executable format.

III. RESULTS

Although the *phpHMM* tool is used to generate the entire speech recogniser, some examples of using the generated codes with the result of faster calculations are discussed.

MFCC Optimisations

One of the optimisation methods is to calculate the results in advance, provided that all operands are

known at compile-time. This will avoid repeated counting still the same results in the recognition process and it speeds up the calculation.

This method of so-called "lookup table" we used to generate the Hamming window coefficients, which are calculated at the beginning of the signal parametrisation by the mel-cepstral coefficients (MFCC), where speech attributes are extracted from the input signal. The parametrisation method during recognition process does not change, and therefore the Hamming window coefficients do not change. The calculation then reduces to reading the coefficient in the one-dimensional data field.

Another part of parametrisation block of signal, is the calculation of Discrete Cosine Transform (DCT). Using the standard method of calculating the DCT, which calculated with goniometrical functions, at the (OMAPtested digital signal processor L137 @ 300MHz, DSP C674x core only) the calculation time of the parametrisation approximately 55 ms per segment was achieved. With the known number of DCT coefficients, which are the constants known at compile-time and during the recognition do not change, the concrete cosine results are calculated in advance and stored to the data structure. When running the DCT algorithm in real-time, then (paradoxically) the cosine is not calculated, but the pre-calculated cosine value is used according to the appropriate arguments. The calculation of the coefficient is thus reduced to reading its value from the pre-calculated table. By this optimisation we reached the calculation time of less than 6 ms, thus approximately 9-times acceleration.



Fig. 2: Computation time vs. optimisation methods for MFCC

Output Probability Density Optimisations

Some of our proposed optimisation methods use transformed parameters, which arise by converting the original model parameters. Eg. a modified algorithm for calculating the output probability function b(o), which is based on the type of $A=A+B\times C$ operation (dot product, "Multiply and Accumulate") [3], requires the recalculation of the original coefficients by a simple transformation. This transformation is performed just during the generation of the source codes. The calculation without optimisations lasted on the dual-core OMAP-L137 DSP 1477 ms/segment, while after the application of appropriate optimisations by recomputing the data structures and modifying the algorithm, the best time 52 ms/segment was reached.



Fig. 3: Computation time vs. optimisation methods of b(o) function

Viterbi Algorithm Optimisation

A part of the Viterbi algorithm, which evaluates the most probable passage through the model, is to compare of any two adjacent values in the vector of results of previous operations. Various methods were tested, but the "Loop Unroll" method proved to be the fastest in this case on the OMAP-L137 hardware platform. The code, that was originally performed repeatedly in the cycle, is broken down into multiple particular operations without the cycle loop. This will both reduce the overhead of cycle organisation, but mainly there will be possibility of greater use of the hardware architecture. In our case, instead of 32 passes through the cycle, a sequence of 32 individual operations with directly addressed operands was generated. This loop unrolling led to the possibility of use of "MAX2" instruction of C674x platform, which is a SIMD instruction that simultaneously compares two pairs of 16-bit operands and returns two results. The figure below shows the effectiveness of this optimisation for different number of test vectors compared with the best time achieved without use of the loop unroll method.



Fig. 4: Computation time of maximum of neighboring values

IV. CONCLUSION

Our intention is to create a miniature speech recogniser module of Czech language commands, which would be easily configurable through a web interface. The first step towards this goal is our

phpHMM software tool that we use for developing speech recognition algorithms, focusing on applications of digital signal processors and more powerful microcontrollers. The advantage of this tool is easy comparison of optimisation methods, easy to change parameters and user-friendly graphical interface. We use it for compiling source code and data structures tailored to the application. In the future we plan to expand to other hardware platforms (dsPIC and MIPS), and automate other steps, including automatic selection of hardware platforms, depending on the number of recognised words.

ACKNOWLEDGMENT

The research was supported by grants GAČR 102/08/0707 "Speech Recognition under Real-World Conditions", GAČR 102/08/H008 "Analysis and modelling biomedical and speech signals", and by research activity MSM 6840770014 "Perspective Informative and Communications Technicalities Research".

REFERENCES

[1] Young Steve. et al., *The HTK Book.* cambridge University Engineering Department, 2006. [online] URL:

<http://htk.eng.cam.ac.uk/ftp/software/htkbook.pdf.zi p>.

[2] PHP [online]. 2011 [cit. 2011-03-12]. URL: <<u>http://www.php.net/</u>>.

[3] MySQL. The world's most popular open source database [online]. 2011 [cit. 2011-03-12]. URL: <<u>http://www.mysql.com/</u>>.

[4] KREJČÍ, R.: Optimisation of Computationally Intensive Part of Speech Recognizer. In 19th Czech-German Workshop on Speech Processing [CD-ROM]. Praha: Institute of Photonics and Electronics AS CR, 2009, p. 22-26. ISBN 978-80-86269-18-4.

[5] OMAP-L138 Low-Power Applications Processor [online]. 2011 [cit. 2011-03-12]. URL: <focus.ti.com/lit/ds/sprs586b/sprs586b.pdf>

[6] TMS320C5505 Fixed-Point Digital Signal Processor [online]. 2011 [cit. 2011-03-12]. URL: <focus.ti.com/lit/ds/symlink/tms320c5505.pdf>

[7] CortexTM-M3 Technical Reference Manual [online]. 2011 [cit. 2011-03-12]. URL: <http://infocenter.arm.com/help/topic/com.arm.doc.dd i0337i/DDI0337I_cortexm3_r2p1_trm.pdf>

[8] KREJČÍ, R.: Use PHP preprocessor for generating source codes in C programming language. In Králíky 2010. Brno: Brno University of Technology, 2010, p. 84-87. ISBN 978-80-214-4139-2.