# Hindi Vocalizer on Heterogeneous Multicore Architecture

April 26, 2013

Gaurangi Anand
anand.gaurangi@iiitb.org

Jayati Deshmukh
jayati.deshmukh@iiitb.org

N Puneeth
puneeth.n@iiitb.org

Pushpendra Sinha
pushpendra.sinha@iiitb.org

Sindhu Priyadarshini
sindhu.priyadarshini@iiitb.org

**International Institute of Information Technology, Bangalore**

**Technical Reports IIITB-TR-2013-10C**

## Abstract

A vocalizer is a system that reads out aloud any written input provided to it. It has multiple applications, particularly in a country like India where the literacy rate is low. Thus, there is an acute need for a Hindi Vocalizer. In this project, a Hindi vocalizer has been implemented on the heterogeneous multicore architecture of Cell Broadband Engine (CBE). This project is novel because of its implementation on such an architecture. It takes the input from the user in the form of text in Devanagari script and gives the corresponding audio file as its output. CBE has a master-slave multicore architecture. The process begins with the user's input to the master core i.e. the PPE. The input is in the form of Devanagari text. The PPE maps this Devanagari text to English syllables based on a predefined set of mappings. Now this mapped input text is distributed among the various slave cores i.e. the SPEs. SPEs map the text given to it with the corresponding audio files associated with it. Each of the SPEs concatenate the different audio files called the phonemes, to create one output audio file for the segment of text each of the SPEs received. The PPE collects the outputs generated by all the SPEs and concatenates them to create single audio file for the input text. This way the system vocalizes the Hindi text exploiting the heterogeneous parallelism feature provided by the CBE.

**Project URL**: https://github.com/textvocalizer/10c

## ***Acknowledgement***

We would like to extend our deepest regards to our advisor, ***Prof. Shrisha Rao*** for his motivating words and useful suggestions throughout this work. It was a pleasure for us to work under his guidance. We are very grateful to all the people who have helped and supported us during the project. We would also like to thank ***Mr.Arun K*** for his guidance.

# Contents

## List of Figures

# 1 Introduction

## Objective

The main objective of the project is to develop a Text-to-speech (TTS) system for Hindi language. The system has been implemented on a heterogeneous multicore architecture. Such an asymmetric architecture is provided by Cell Broadband Engine which is present in the Sony's Playstation 3.

## Motivation

Hindi, the official language of India, is spoken as a first language by 33 percent of the Indian population, and by many more as a *lingua franca*. In contrast, a very small percentage of Indians uses English as a means of communication. Coupled with the prevalent low literacy rates, the use of conventional user interfaces that use English, are difficult to use and understand by the common man in India. Spoken language interfaces enabled with TTS synthesis have the potential to make information and other Information and Communication Technology (ICT) based services accessible to a large portion of the population.
TTS system is the most widely used system in speech technology. The various TTS synthesizer systems that are available are Festival, Multilingual, etc. It is a computer-based system that should be able to read aloud any text, whether it was directly introduced by a user or scanned and submitted through an Optical Character Recognition system. Speech synthesis is a process where verbal communication is replicated through an artificial device. There is a need for a viable TTS system that can be used in a commercial and institutional setting.

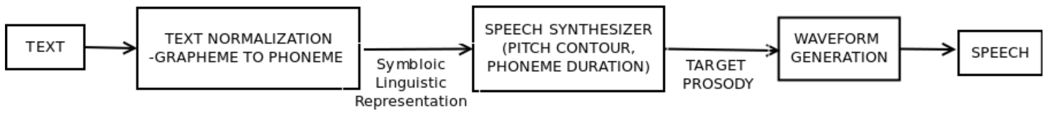
TEXT
TEXT NORMALIZATION -GRAPHEME TO PHONEME
Symbloic Linguistic Representation
SPEECH SYNTHESIZER (PITCH CONTOUR, PHONEME DURATION)
TARGET PROSODY
WAVEFORM GENERATION
SPEECH

Figure 1: A Text-to-Speech System

## 2 Overview

### 2.1 Vocalizer

A vocalizer is a TTS system that has to be sophisticated enough to recognize the piece of text accurately and read it aloud. The system that analyses the text, processes it, generates speech for corresponding words processed and produces voice is a TTS system as shown in Figure 1. The Natural Language Processing (NLP) engine has to be robust enough to generate the sound for the words interpreted as they are meant to be pronounced. The database used for recognition is vast and includes all possible characters used in the Hindi language. The goal of TTS is the automatic conversion of input in text form to a spoken form that closely resembles the speech of a native speaker of that language. This field of speech research has witnessed significant advances over the past decade with many systems being able to generate a close to natural sounding synthetic speech. Research in the area of speech synthesis has been fueled by the growing importance of many new applications. These include information retrieval services over telephone such as banking, public announcements at places like train stations, bus stands and reading out of manuscripts [1]. Speech synthesis can also be applied in tools for reading emails, faxes and web pages over telephone and voice output in automatic translation systems or to improve the accessibility.

## 2.2 Cell Broadband Engine

Cell is a heterogeneous multicore processor comprised of control-intensive processor called the Power Processing Element (PPE) and compute-intensive SIMD processor cores called Synergistic Processing Element (SPEs) [2, 3] as shown in Figure 2, each with their own distinguished features. A high-speed bus called the Element Interconnect Bus (EIB) is used for connecting these processor cores within the Cell. The EIB also provides connections to main memory and external I/O devices, making it possible for processor cores to access data and interact.

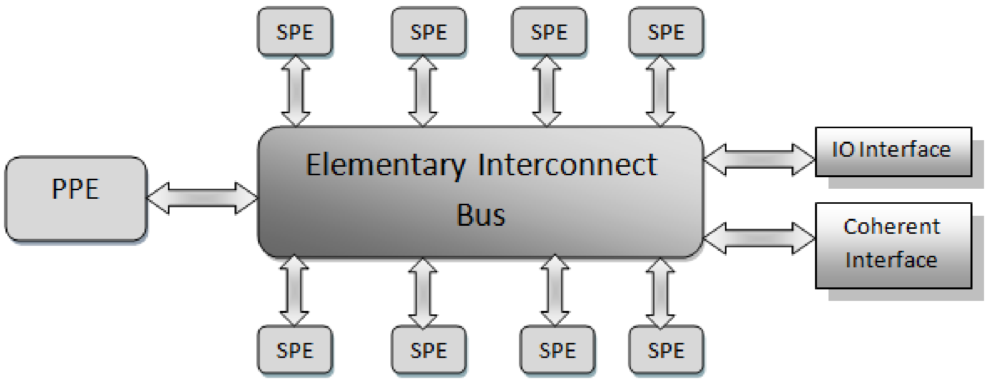

Figure 2: Basic CBE Architecture

Basically cell has 2 types of cores :

- **PowerPC Processor Element (PPE):** The PPE implemented in the Cell is a general-purpose processor with functions equivalent to those offered by the 64-bit PowerPC architecture. PPE allows execution of the operating system and applications. It also performs input/output control when the operating system accesses the main memory and external devices, and provides control over SPEs. It acts as master core that controls all the SPEs. They are designed as a control-intensive processor dedicated mainly to processing control.

- **Synergistic Processing Element (SPE):** The Cell incorporates eight compute-intensive processor cores called the SPEs. They are less complex processing units than the PPE as they are not designed to perform control-intensive tasks. They iterate simple operations necessary for processing multimedia data. The Cell delivers an exceptional computational capability by selectively using these computationally intensive processor cores. These SPEs are the primary computing engines on the Cell processor. Each of them contains a Synergistic Processing Unit (SPU), a memory flow controller, a memory management unit, a bus interface and an atomic unit for the synchronization mechanisms. To make full use of all the computational power available on the Cell BE processor, data must be distributed and communicated between PPE and SPEs. The PPE interacts with the SPEs through Memory-Mapped Input/Output (MMIO) registers supported by the Memory Flow Controller (MFC) of each SPE. PPE is often used as an application manager, assigning and directing work to the SPEs. A large part of this task is loading main storage with the data to be processed, and then notifying the SPEs. The SPEs do all the processing. When the SPEs are done with the operations they were assigned, the PPE is notified to collect data and generate the output in the desired format.

## 2.3 Description

This is a Text-To-Speech system in which the input is given in the text form written in an Indian language. Hindi has been chosen as the language for which the written text in Unicode format, in the Devanagari script, will be vocalized in the pseudo human voice. In this project, the text input is accepted by the PPE. It carries out initial processing and then it distributes the data among all the SPEs that are compute-intensive, hence iterate operations on the data they receive. Each SPE carries out their part of the task independent of what other SPEs do. When all the SPEs which work in parallel are done with their jobs, the processed data is made available to the PPE from all the SPEs. The PPE then generates the final output i.e.

the audio output file for the complete text. This audio output file is the vocalized form of the text that was given to the system as input.

**Rule-based approach**

The approach used for vocalizing is Rule-based. In this approach, each word from the sentences of the input text is concentrated upon. Each word composed of consonants, vowels or their combination, is called phoneme. The database consists of all the phonemes that form the part of the Hindi language. Each word is then split into consonants and vowels, and then depending upon the spellings, the consonants are concatenated with the vowel following them and these form the various phonemes. These phonemes are fetched from the database and are combined together for each of the sentences of the segments each SPE receive. This approach is called rule-based as the speech output for whole of the text is generated based on the spellings of each of the words of the sentences in the input. After all the SPEs have generated the audio file for all the sentences of their respective segments, they notify the PPE. The PPE then combines all the outputs of all the SPEs taking into the consideration, the order in which each SPE was allocated a segment of the text. Then the PPE plays the speech generated for the original text that was presented as input to the system. This approach is unlike the dictionary based approach in which all words in the language have their corresponding sound files in the database. Rule based approach brings in the flexibility to fetch the sound files for any word in the text whereas it is computationally intensive in the dictionary-based approach. It may not be feasible to have all sound files for all the words in the Hindi dictionary in the database and this process would waste a lot of resources, both in storing sound files in the database as well as in fetching them.

**Parallel processing**

The central core, i.e. the PPE accepts the input. It then tokenizes the input and splits the whole text into various segments that contain complete sentences, to be distributed among all the SPEs equally. These segments contain the tokenized strings and each segment is read by individual SPE that reads and generates corresponding audio file for the segment it received. This processing of the segments is carried independently by each of the SPE that help us attain thread level parallelism where each thread manages each of the SPEs. This way the feature of multicore heterogeneous parallelism is exploited.

## 3 Gap Analysis

**Similar System** Various Many Text-To-Speech systems are available and various are being worked upon. But there are a very few for Indian languages, especially Hindi. There are 2 most common Text-To-Speech systems available, namely, Dhvani and Vani.

- Dhvani
  It is a Text-To-Speech system developed for Indian languages[6]. It consists of speech synthesizing module that makes use of voice database in the Indian languages. It was started as a project by Dr. Ramesh Hariharan, Indian Institute of Science, Bangalore in 2001. Then it was taken forward by Mr. Santhosh Thottingal in 2006, who rewrote many parts and added more modules to it. Presently, he is maintaining the project. It is written entirely in C. It has a command line interface as well as a C/C++ API interface. It has a database of sound files of about 2MB size with the feature of pitch modification available. It can even be used with Python using the python speech binding. The text is mapped with the phonetic description that is syllable based.

- Vani
  It is a Text-To-Speech system developed as a project by Computer Science Department, Indian Institute of Technology, Mumbai[7]. The motivation for its development was to get the software speak exactly what the listener wants. It is not only based on the simple concatenation synthesis but also uses signal processing that brings in flexibility for how the audio is being generated. It can express emotions and can

also sing if desired. It makes use of extensive language processing with which any word can be expressed differently with options of various expressions and tones available. It has been developed on Java and makes use of Java sound API.

Good quality Hindi or any local language vocalizer system that can be used for practical applications are presently not available. Though a number of prototypes of Indian language TTS systems have been developed [4, 5], none of these can be compared to the systems in languages like English, German and French which have attracted a lot of research and development. The main reason for this is that developing a TTS system in any new language requires close collaboration between linguists and technologists for solving many language specific issues. Significant amount of annotated data is required for developing language-processing applications which involves many parts such as speech (POS) taggers, syntactic parsers, intonation and duration models. This is a resource intensive task, which requires significant institutional support which developing countries generally lack.

## 4 Architecture

The whole system architecture and the flow of the project is shown in Figure 3. The complete flow of the project is deployed in the CBE.
The working of the system starts with the input which is in the form of user query. which is a text file containing the Hindi text in Devanagari script in UTF-16 encoding. The process begins with the input to the PPE. PPE now maps this Devanagari text to English, based on predefined mapping rules. Now this mapped input text is distributed among the various SPEs. SPEs map the text given to it to the corresponding speech/audio files associated with them by fetching their respective phonemes from the database. Each of the SPEs concatenate the different audio files to create one output audio file for each of them. All the different audio files are further concatenated by the

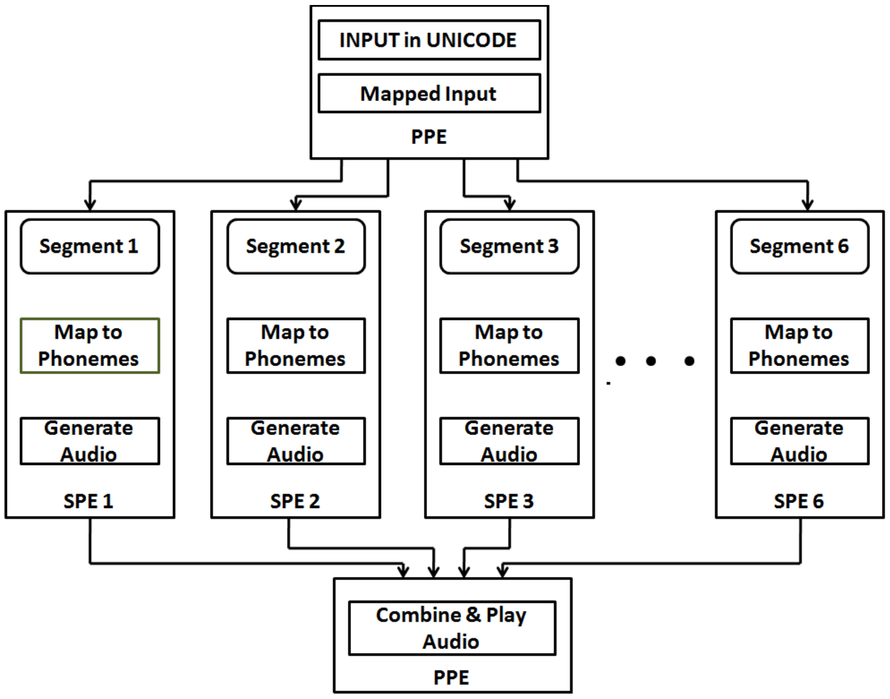

Figure 3: Complete System Architecture

PPE to create one final output file for whole of the text that was provided as input to the system.

The architecture can be described in a nutshell as follows:

- **Input Preprocessor:**
  In the pre-processing step all the special symbols are removed from the given Devanagari text .

- **Mapping of Input:**
  In this step the pre-processed Devanagari text is parsed and each of the Hindi character is mapped to the corresponding English alphabets for the consonants and integers for vowels.

- **Mapping to Phonemes:**
  The English alphabets and integers (if present) are joined together so as to convey proper phonetics.

- **Generation of Audio Files:**
  The phonemes so formed are converted into the audio files by fetching from the database.
- **Concatenation of Audio Files:**
  Audio files generated by different SPEs are then combined to form the final audio file and played by the PPE.

# 5 Implementation

## 5.1 Modules of TTS

#### Input to Unicode

Input is given by the user in Hindi in Devanagari script in a text file. The user can give multiple sentences as input in the input file. UTF-16 has been used as the Unicode standard for the input.

#### Mapping of Input

Firstly, the input file is read and split into characters. An array contains the stream of characters of the input. A file contains the mapping for each character in Hindi to the corresponding mapping in English like ka, kha and so on. Every consonant is mapped to a string in English and every vowel is mapped to an integer. Further processing is done on this mapped input. The input is now partitioned into segments to be further processed by the SPEs in parallel.

#### Mapping to phonemes

Now the array containing the string in English is processed. Since a vowel can occur only after a consonant so the processed input string is modified such that every integer is preceded by a string in English. Next, the input which now contains a set of strings, which may or may not be followed by an integer, are mapped to phonemes like ka.wav, kha.wav, ma5.wav and so on. These phonemes are already stored in the database of the system.

#### Generation of Audio

All the SPEs merge the phonemes as per the processed input, each of them working in parallel on the segment they received. Finally, an output file is generated by each SPE.

#### Combine and Play audio

The output files of all the SPEs are combined into one final output file which contains the speech for the input. This combined output file is a .wav file that is played by the PPE for the user, keeping into the consideration, the order of the segments that were allotted to each of the SPEs.

#### Database of sound files

The database is created as per the number of alphabets in the Hindi language. It contains around 400 sound files. These files have been recorded using Audacity tool. A 2 stereo input channel has been used for high quality of sound recording. 44100 Hz is the project rate of sound files. Average length of the sound files is about 0.430 seconds.

## 5.2 Choice of Technology

Java and C has been used as the programming languages for development of the TTS System for Hindi. Java program runs on the PPE that maps the input to English as it is easier to handle Unicode in Java in comparison to C. Also tokenization into characters is better to process in Java in comparison to C. Next, the processed input is divided into segments. The SPEs support only C/C++, so further processing and generation of sound files for each segment is done in C on each of the SPEs. Finally the complete output is played by the PPE by executing a shell script from C language.

## Tools Used:

### Audacity:

Audacity is a free, easy-to-use and multilingual audio editor and recorder for Windows, Mac OS X, GNU/Linux and other operating systems.
Audacity can be used to:

- Record live audio, Convert tapes and records into digital recordings or CDs.
- Edit Ogg Vorbis, MP3, WAV or AIFF sound files.
- Cut, copy, splice or mix sounds together.
- Change the speed or pitch of a recording.

### lib-snd:

The libsndfile is a C library for reading and writing files containing sampled sound (such as MS Windows WAV and the Apple/SGI AIFF format) through one standard library interface. It is released in source code format under the Gnu Lesser General Public License.
The library was written to compile and run on a Linux system but should compile and run on any Unix system (including MacOS X). There are also pre-compiled binaries available for 32 and 64 bit windows.
It was designed to handle both little-endian (such as WAV) and big-endian (such as AIFF) data, and to compile and run correctly on little-endian (such as Intel and DEC/Compaq Alpha) processor systems as well as big-endian processor systems such as Motorola 68k, Power PC, MIPS and Sparc. Design of the library also makes it easy to extend for reading and writing new sound file formats.

## 6 Features

- **Multicore**
  The implementation of the Hindi Vocalizer on a heterogeneous multicore architecture has not being tried before. Thus, this is a unique implementation of a TTS system.

- **Data Parallelism**
  The paradigm of data parallelism has been used to implement this system. Data is divided among the various cores which in turn gives an audio output in return.

- **Speed-Up**
  The system is observed to give a speed up as compared to the Intel core i5 architecture. At lower input sizes, the performance of the i5 machine was better than the CBE, but with the increasing input size the performance of CBE was much better.

- **Platform Independent**
  The modular system of implementation has enabled to reproduce this system on x86 architecture without any significant change in the implementation code.

## 7 Testing and Performance

### 7.1 Testing

The system has been tested on multiple input sizes on both the x86 machine and the CBE. The input size of the sentences was varied from 4 to 8 words per sentence.
The number of sentences which the input file contained varied from one to four hundred sentences.
A wide variety of words and sentence constructions has been included in the input text, in order to exploit the database to the fullest.

## 7.2 Performance

### 7.2.1 x86 vs. CBE

The performance of CBE can be compared with that of x86 architecture depending on the size of the input given to each of them. From the figure, it can be seen, for smaller inputs the performance of x86 is better. The time taken by it is less than the time taken by the CBE to process the input text to generate corresponding audio files. The performance graph shows that initially the two lines run almost parallel to each other with the CBE taking more time. At input size of about 220 sentences, both the architectures take the same time. But as soon as the input size is increased beyond 220 sentences, the performance of CBE boosts up and the x86 architecture starts taking more time leading to increase in the slope of the performance of the latter.

This can be seen explicitly in figure 4.

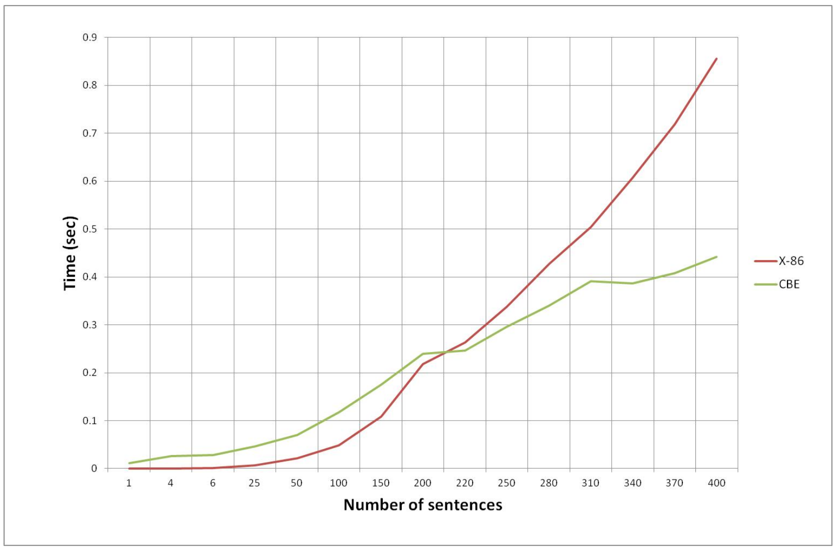

Figure 4: Performance comparison of CBE vs. x86

### 7.2.2 Degree of Parallelism

The performance of the system was checked with varying degree of parallelism. The system was tested with bounds on the number of SPEs for multiple sentences.
There were three test runs, one each with 200, 300 and 400 sentences. To avoid any arbitrary bias or error, a total of 100 sample readings were taken for each run and the average of those samples were plotted on the chart.
The chart clearly depicts the effect of parallelism on the performance of the system.
With increasing degree of parallelism, the time taken by the system monotonically decreases. Hence, the degree of parallelism and time taken are inversely proportional.
Figure 5 summarizes the results of this test.

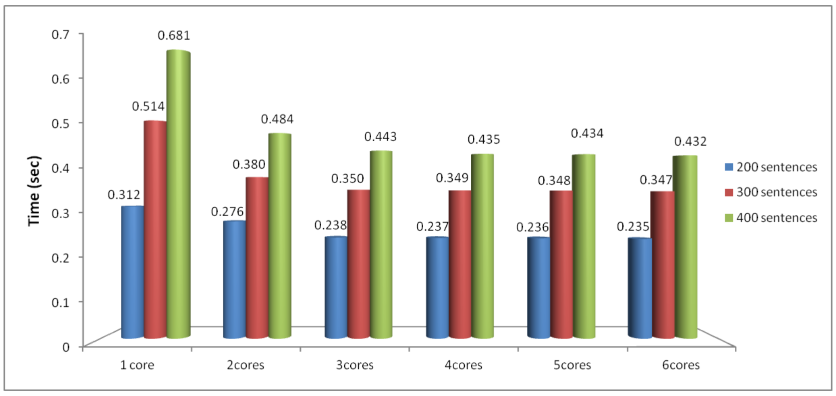

Figure 5: Performance analysis for varying degree of parallelism

# 8 Conclusion and Future Work

This TTS system is suitable where there is a need for a large throughput. It could be used in a variety of applications such as in the telephone network which has to deal with millions of phone calls, in the public announcement systems, in institutions serving the visually impaired etc.
The increasing prevalence of multicore systems both heterogeneous and homogeneous to deal with the problem of scalability means that the system can easily be implemented on any similar architecture.
The project primarily focuses on the implementation of the TTS system over the heterogeneous multicore architecture.
Further work can be done with the processing of the sound waves of the speech to bring in variations. The prosodic feature of the speech needs to be further improved so that pitch, intonation and other features of the output can be as close to real human voice. This will further boost the commercial viability of the system.

## References

[1] Samuel Thomas, "Natural sounding speech synthesis based on syllable-like units", M.S. thesis, Department of Computer Science and Engineering, IIT Madras, India, May 2007.

[2] Sony Computer Entertainment. (2008,August) *Guide to Cell Broadband Engine Programming Documentation.* (version 1.02). [Online]. Available: `http://cell.scei.co.jp/pdf/CBE_Architecture_v102.pdf`

[3] Yang Song, Gregory M. Striemer, Ali Akoglu, "Performance Analysis of IBM Cell Broadband Engine on Sequence Alignment," *2009 NASA/ESA Conference on Adaptive Hardware and Systems*, pp.439-446, 2009

[4] S.P. Kishore, Rohit Kumar, and Rajeev Sangal,"A Data-Driven Synthesis approach for Indian Languages using Syllable as Basic Unit", *International Conference on Natural Language Processing (ICON)*, 2002, pp. 311 to 316.

[5] J. Rama, A.G. Ramakrishnan, and R. Muralishankar, "A Complete TTS system in Tamil", *IEEE Workshop on Speech Synthesis*, 2002.

[6] Dr Ramesh Hariharan, Santhosh Thottingal, *Dhvani, TTS system for Indian languages*. [Online]. Available : `http://dhvani.sourceforge.net` [Feb. 10, 2013]

[7] Harsh Jain, Varun Kanade, Kartik Desikan, *Vani, an Indian language text to speech synthesizer*. [Online]. Available : `http://vani.sourceforge.net` [Feb. 15, 2013]