T2R2 東京科学大学 リサーチリポジトリ
Science Tokyo Research Repository

論文 / 著書情報
Article / Book Information

論題(和文)	
Title(English)	Recent Functionality Improvements to the T3 Speech Decoder
著者(和文)	DIXON PAUL RICHARD, 大西 翼, 古井 貞熙
Authors(English)	Paul R. Dixon, Tasuku Oonishi, Sadaoki Furui
出典(和文)	, , No. 3-1-10, pp. 113-114
Citation(English)	日本音響学会2009年秋季講演論文集, , No. 3-1-10, pp. 113-114
発行日 / Pub. date	2009, 9

Powered by T2R2 (Science Tokyo Research Repository)

Recent Functionality Improvements to the T^3 Speech Decoder*

◎ Paul R. Dixon Tasuku Oonishi Sadaoki Furui
Tokyo Institute of Technology, Tokyo, Japan
{dixonp}@furui.cs.titech.ac.jp

1 Introduction

This paper describes some of the recent and current development work to the T^3 (t-cubed) speech recognition decoder. These include better hardware and platform support, improved multicore acoustic scoring and a grapheme to phoneme system added to the supporting tools.

2 Platforms and Architectures

The fundamental goals of the core engine are speed, performance and resource usage. Recently we have focused on portability and platform support. Developing a highly scalable single code base which can make use of wide range of hardware is a significant challenge, in particular the engine should be able to scale up and down with the target hardware. The recent efforts to realize this aim are as follows:

- To support Graphics Processing Units (GPUs) and parallel processors from multiple vendors. A layered approach has been added to allow different device specific Application Programming Interfaces (APIs) to be evaluated more easily.
- For the PowerPC and other big endian processors the byte ordering is automatically handled internally to allow binary compatibility with the standard x86 byte ordered models.
- The decoder has been compiled and tested on Unix, Windows and MacOS platforms and on x86, AMD64, PowerPC and ARM (via an emulator) processors.

2.1 Layered Acoustic Scoring

A unified abstraction layer has been implemented to encapsulate the GPGPU scheme we proposed in [4]. Underneath this layer various *device* layers are used to perform the actual computation. The motivation is to allow for different device layers to be implemented around the various GPUs and data parallel APIs such as OpenCL or DirectX compute shaders. Device layers are not just restricted to GPUs, alternatively they could be the Synergistic Processing Units (SPUs) on the Cell processor, a conventional multicore processor or a totally different class of a device such as a Field Programmable Gate Array (FPGA).

The abstraction layer provides the high level interfaces needed by the decoder to create the matrices of acoustic parameters and launch the transfers or computations. A Synchronization barrier is also available to allow the acoustic computations to be performed in an asynchronous fashion. In this overlapped mode the upcoming acoustic scores are computed whilst the search algorithm operates on the current window of acoustic scores.

2.2 High-level API

Previously there were two levels at which to interact with the T^3 decoder. By shell level executables or at the source level in C++. To make the decoder more accessible to researchers or developers and easier to embed into application we are creating a set of Windows .net APIs. These high level interfaces follow the method signatures of the Microsoft Speech API.

2.3 Trainable Grapheme-to-Phoneme Module

A new tool available is a WFST based grapheme-to-phoneme (GTP) module for learning and generating pronunciations. The system is fully trainable from an unaligned pronunciation dictionary and places no restrictions on the language. The tool first generates a set of uni-gram alignments from the pronunciation dictionary. The alignments are then used to build the G2P WFST using a procedure similar to the scheme described in [3]. The alignment pairs are converted into a sequence of the form $\langle g,p\rangle_1, \langle g,p\rangle_2, \ldots \langle g,p\rangle_k$, where g is grapheme and p is a phoneme. The entire set of sequences is used to a train an n-gram language model.

Next the n-gram language model is converted to an n-gram weighted acceptor representation where each input label belongs to the set of transliteration alignment pairs. Next the pairs labels are broken down into the input and output parts and the acceptor is converted to a transducer M. To perform the actual G2P conversion the input word is converted to an acceptor I which has one arc for each of the characters in the word. I and M are combined according to $I \circ M = O$ where $\circ$ denotes the composition operator. The n–best paths are extracted from O by projecting the output, removing the epsilon

*T3 音声認識デコーダの最近の機能改良
ディクソン・ポール、 大西 翼、 古井 貞熙

	Baseline		Multicore			Multicore Overlapped			GPU Overlapped		
Beam	ACC	RTF	ACC	RTF	SU	ACC	RTF	SU	ACC	RTF	SU
125	77.65	0.351	77.65	0.349	1.01	77.65	0.345	1.02	77.78	0.066	5.32
150	79.42	0.664	79.42	0.449	1.48	79.42	0.406	1.64	79.69	0.151	4.4
175	79.93	1.1	79.93	0.649	1.69	79.93	0.572	1.92	80.15	0.335	3.28
200	80.06	1.634	80.06	0.934	1.75	80.06	0.871	1.88	80.31	0.631	2.59

Table 1: Comparison of the multicore acoustic scoring implementations in comparison to the vectorized CPU and GPU implementations. ACC is the word accuracy, RTF real time factor and SU is the speed-up over the SSE baseline.

labels and applying the n-shortest paths algorithm with determination from the OpenFst Toolkit [1].

3 Evaluations

In these evaluations we demonstrate a device layer that uses a standard multicore processor and is able to substantially speed-up the decoder. To perform the matrix multiplication we used the AMD *ACML* library[2] which is freely available and even though aimed at AMD processors, it can also achieve very high performance on Intel processors.

We compared the multicore acoustic computations in both synchronous and overlapped modes to our standard on-demand vectorized CPU implementation and GPU implementation programming through Compute Unified Device Architecture (CUDA). The CPU was an Intel Core 2 processor with four cores and the GPU was a CUDA GTX280.

The recognition task used 2328 utterances from the Corpus of Spontaneous Japanese. The acoustic features were 39 dimensional MFCCs, the acoustic models had 3000 states each with 128 Gaussians and the search network was built from a tri-gram language model to give a final size of 1.1M arcs and 2.4 states. The band was set at 10000 and the beam width varied from 125 to 200, the window size used in the acoustic computations was 64 samples for all devices.

Table 1 shows the performance of the various acoustic scoring implementations, we see the GPU device is fastest in all cases and obtaining higher accuracies due to the use of a full logsum in the likelihood computations. By using the synchronous multicore acceleration in the best case we obtained a 1.75 times speed-up. By operating in overlapped mode it was possible to obtain over further 10% improvement in speed and this allows the multicore approach to nearly half the decoding time. One of the key points of our implementation is no low-level programming is required because we create a simple wrapper around highly optimized libraries from the hardware vendors, and furthermore it is very simple to rapidly evaluate different device layers.

We also performed evaluations on the Cell processor by utilizing a PlayStation 3 console with Linux installed. These experiments confirms the decoder can run on big endian hardware and with binary comparability. However, these initial evaluations show the decoder running several times slower than the current x86/AMD 64 version. Future research is needed to fully understand how to obtain maximum performance on this platform. It could be our choice of hardware or software platform, for example performance could be improved by using a version of the operating system with huge page support. These initial evaluations illustrate it is essential to fully utilize the SPUs to obtain best performance on the Cell based processors.

4 Conclusions and Future Work

In this paper we have described some recent and current additions to our decoder and toolkit. In particular we have shown our latest multicore implementation can nearly halve the decoding time.

When a production ready OpenCL implementations becomes available we would like to evaluate OpenCL implementations of our GPGPU acoustic scoring on various different GPUs. OpenCL is industry standard API for targeting GPUs and parallel processors and should allow us to support a greater range of hardware from a smaller code base. Another area we are investigating is moving the decoder to embedded and computationally limited platforms.

References

[1] C. Allauzen, M Riley, J. Schalkwyk, W. Skut, and M. Mohri. OpenFst: A general and efficient weighted finite-state transducer library. In *Proc. of the Ninth International Conference on Implementation and Application of Automata, (CIAA 2007)*, pages 11–23, 2007.

[2] AMD Corporation. `www.amd.com/acml`.

[3] D. Caseiro, I. Trancoso, L. Oliveira, and C. Viana. Grapheme-to-phone using finite state transducers. In *In Proc. 2002 IEEE Workshop on Speech Synthesis*, 2002.

[4] P.R. Dixon, T. Oonishi, and S. Furui. Harnessing graphics processors for the fast computation of acoustic likelihoods in speech recognition. *Computer Speech and Language*, 23(4):510–526, 2009.