論文 / 著書情報
Article / Book Information

Title	Recent Development of WFST-Based Speech Recognition Decoder
Author	Paul Dixon, Tasuku Oonishi, Koji Iwano, Sadaoki Furui
Journal/Book name	Asia-Pacific Signal and Information Processing Association 2009 Annual Summit and Conference, , , pp. 138-147
Issue date	2009, 10

Recent Development of WFST-Based Speech Recognition Decoder

Paul R. Dixon *, Tasuku Oonishi *, Koji Iwano† and Sadaoki Furui *
* Department of Computer Science, Tokyo Institute of Technology
2-12-1, Ookayama, Meguro-ku, Tokyo, Japan, 152-8552
E-mail: oonishi,dixonp@furui.cs.titech.ac.jp,furui@cs.titech.ac.jp Tel: +81-3-5734-3480
† Faculty of Environmental and Information Studies, Tokyo City University
3-3-1 Ushikubo-nishi, Tsuzuki-ku, Yokohama, Japan, 224-8551
E-mail: iwano@tcu.ac.jp Tel: +81-45-910-2598

***Abstract*—In this paper we present an overview of the Tokyo Tech Transducer-based Decoder T^3 (pronounced *tee-cubed*). There is a high level overview of the engine's design and features which is accompanied by a more detailed description of the features that are unique to our engine. These include the ability to perform acoustic computations on a graphics card and generalized fast *on-the-fly* composition and optimization algorithms. We describe voice activity detection functionality recently added to the engine and finally results are presented which show the engine achieving very high recognition throughput at a high recognition accuracy.**

I. Introduction

The T^3 (pronounced *tee-cubed*) speech recognition engine is a modern state-of-the-art speech decoder that is currently under development at Tokyo Institute of Technology. T^3 began with the goals of developing a recognition engine which would not only provide high performance in terms of accuracy, speed and resource usage but also a robust foundation for use in practical applications and as a research platform for implementing and exploring new techniques.

The engine is designed to operate on Weighted Finite State Transducers (WFSTs) [25] which are a type of finite state machine that can provide a mapping between strings with an optional weight to represent uncertainty. Recently, the use of WFSTs in speech recognition has become extremely popular, one of the main advantages of the approach is the unified manner all of the models optimized and combined together. Furthermore, performing the optimization ahead of decoding allows for the development of speech recognition engines that can often deliver faster recognition speeds when compared to more traditional dynamic decoders[16].

A speech recognition system will often be part of a much larger spoken language processing system and require additional support tools. For example Grapheme-to-Phoneme conversion for generating pronunciations[7], dialog processing[13], machine translation[21] or speech synthesis[6], [1] to name a few. The general nature of WFSTs allows each of these tasks to be represented in the WFST paradigm bringing further unification of tools and techniques.

However, there are several drawbacks to the unified approach, firstly large amounts of memory can be required to hold the fully composed network during recognition and the off-line memory usage during composition and optimization may become prohibitively large. After the composition access to the information source is lost and therefore changing the models on-line becomes much more difficult. Later in the paper we describe the on-the-fly techniques we have developed to deal with these issues.

In this paper we first provide a broad overview of the design and features of T^3 and describe in detail some of the more important and unique features. During development we focused heavily on resource usage and recognition speed. Two of the important features of the engine are the ability to offload acoustic computations to a Graphics Processor Unit (GPU) enabling extremely fast decoding even on very large acoustic models and novel versions of the WFST algorithms that perform on-line composition and optimization.

The core feature set includes all of the functionality one would expect from a modern recognition engine, such as an integrated front end, lattice output, n-best lists and confidence scores. The decoder was designed with compatibility in mind to allow the use of existing resources and compare performance. For example the decoder and supporting tools can deal with acoustic models in ATT[24] or HTK[33] format and the frontend can generate features that are compatible with HTK/Julius[33], [19] or the CMU Sphinx[18] toolkits. We use the ATT format for the text representation of WFST and this gives compatibility with the ATT FSM[24] and OpenFst[4] toolkits.

To reduce physical memory usage during recognition T^3 can leave the static search network on the disk and pull in the required state and arc information as requested by search. The initial motivation for this technique was to allow for the T^3 to run as an embedded recognition engine for example on a PDA. Since the initial development of T^3 the disk based approach has become more appealing because of the recent proliferation of solid state drives in particular on low-end *netbooks* which are well suited to this approach. In addition T^3 can also perform the composition *on-the-fly* and this allows for the component models to be loaded and used by the decoder directly without having to perform the off-line composition phase.

The memory savings achievable by on-the-fly composition

often come at the cost of a reduction in the speed of the decoder. This is mainly because it is not possible to perform optimization on the fully composed network and the cost of the composition itself. To deal with these issues specialized fast on-the-fly composition and optimization schemes have been proposed by others [14], [8]. In the T^3 decoder we have also developed a generalized fast on-the-fly composition which requires no modification to the decoder similar to the work by [8]. The proposed method can also perform optimization on-line and places no restrictions on the topology of the component networks.

II. Weighted Finite State Transducers

We first start with a brief overview of WFSTs that will cover the theoretical foundation needed to describe the algorithms presented later in the paper. For a more in depth description of WFSTs in speech recognition the reader is referred to [25], [26], [23]

A WFST is a generalized type of finite automata where each of the transitions has an output label and optional weight in addition to the input label. Formally a transducer T is defined as the 8 tuple[26], [23]:

$$T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho) \quad (1)$$

Where:

- Σ is a finite input alphabet.
- Δ is a finite output alphabet.
- Q is a finite set of states.
- $I \subseteq Q$ is the set of initial states.
- $F \subseteq Q$ is the set of final states.
- $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times \mathbb{K} \times Q$ is a finite set of transitions.
- $\lambda : I \mapsto \mathbb{K}$ the initial weight function.
- $\rho : F \mapsto \mathbb{K}$ the final weight function.

A. WFSTs in Speech Recognition

The power in the WFST framework is the way we can represent each of the knowledge sources used in speech recognition in a consistent manner, compose them together and apply the optimization algorithms.

We construct our recognition cascade from the following components; the language model G which represents the recognition grammar, the lexicon L which is built from the pronunciation dictionary and maps phoneme sequences to words, a transducer C that converts context-dependent phonemes to context-independent phonemes, and optionally the acoustic models H.

The full recognition cascade is constructed according to $min\,(det\,(H \circ det\,(C \circ det\,(L \circ G))))$[26], where det is the *determinization* algorithm that performs a prefix sharing of the WFST, and min the *minimization* operation which performs a suffix sharing.

During the search T^3 dynamically expands the arc definitions which are either Hidden Markov Model arcs corresponding to H or factorized sequences of acoustic model states. It is also possible to decode at state level networks by omitting the factorization step and then simulating the self-transitions via an appropriate arc definition that contains a single HMM state.

III. T^3 Design

The architecture of the T^3 decoder is shown in Figure 1. Our decoding engine is not just the search component, but is a fully functioning standalone real-time speech recognition engine[9]. The whole system can be broadly thought of as several phases. A training phase where the language and acoustic models are trained on speech data. Next is the off-line conversion phase where the knowledge sources are then transformed into WFST representations and combined using the method described in section II-A. In the recognition phase the engine is configured with the processed models and then it converts the speech input to the desired output format.

Within the recognition engine there are three main blocks the *frontend*, *decoder* and *control* units. The control unit is the simplest of the blocks, which takes the configuration information and sets up the decoder and front-end. The job of the frontend is to take the raw speech and convert it to a set of feature vectors suitable for recognition. The decoder sub-unit performs the actual searching of the feature vectors on the models to produce the recognition results. The significance in the separation of the decoder and frontend modules is that either of these models can be operated as individual standalone units. This means the decoder can consume pre-computed features directly and the frontend can perform conversion for off-line usage.

A. Frontend

The frontend is designed to be modular and extensible using an architecture similar to the scheme described[18]. The extraction pipeline is constructed from a sequence of smaller *filter* units that perform some transformation on the signal. We provide a repertoire of filters and place no restriction on the length or ordering of the filter chain. The filter library contains all of the operations needed to perform Mel-Frequency Cepstrum Coefficient (MFCC) conversion, including but not limited to the following:

1) Pre-emphasis
2) Windowing
3) Fast Fourier transform
4) Filter bank analysis
5) Discrete cosine transformation
6) Dynamic features
7) Normalization

The frontend is capable of operating in batch or streaming modes either connected to the decoder or standalone mode. The modularity of the design allowed us to incorporate the VAD into the frontend. The speech data is loaded into a *data packet* object and propagated along the filter pipeline. In addition to the signal the data packet contains additional status flags and signals for other elements of the engine.

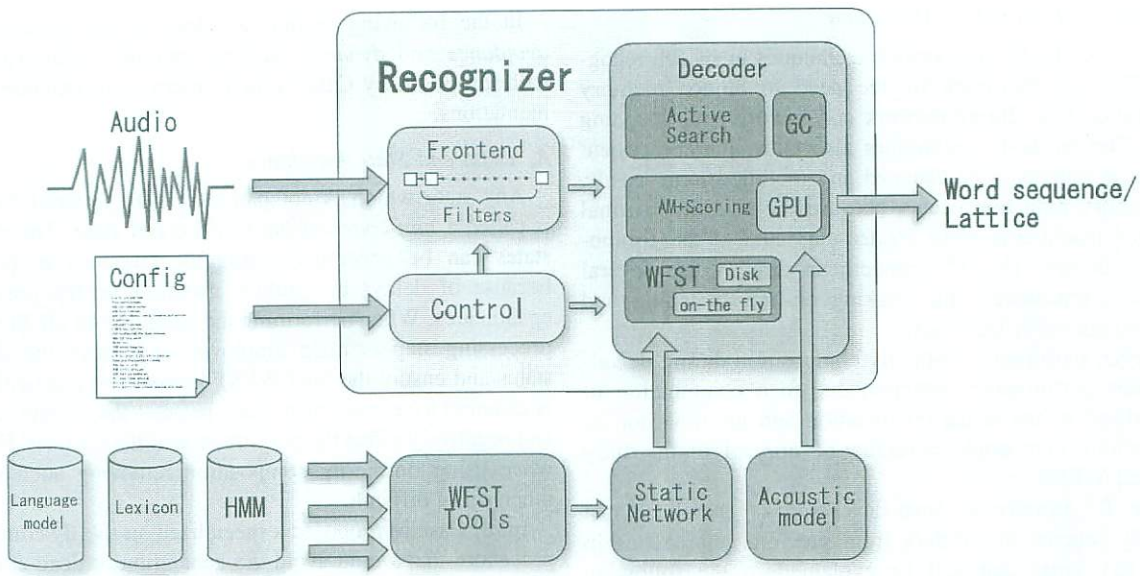

Fig. 1. Architecture of the T^3 recognition engine.

B. Decoder

The decoder is broken down into smaller abstractions where the main blocks includes the following:

- Active search and traceback
- Output processing
- WFST interface and implementation
- Acoustic scoring

1) Search and Traceback: The decoder's core runs the Viterbi algorithm in a time-synchronous manner. When a state is activated from the WFST a set of corresponding search states and search arcs are added to an active lists. Search states or search arcs considered active when they contains a valid token[34]. To decode an input utterance the search algorithm runs the following token passing[34] procedure for each frame:

1) For each active state propagate the token to the initial state of every non-epsilon search arc leaving the state.
2) For each active arc propagate the active tokens one frame. This step uses a specialized time synchronous Viterbi algorithm optimized to the arc topology. If at this point the search arc contains an active token in the last state, the token is propagated to the destination state.
3) For each active state propagate the token over any input epsilon arcs leaving the state.

To contain the size of the search space, beam pruning is applied at several points of each iteration by removing hypotheses that have a cost greater than the combined current best cost plus beam width. Additionally a search band is used to restrict the maximum number of active arcs at any time. Every time a token crosses a phone boundary a record is added to a traceback structure and once the decoding has finished the best path of lattice is obtained by traversing this structure backwards from the best token.

The pruning of hypotheses will create dead paths in the traceback lattice which will not be extended by active tokens. A *mark-sweep* garbage collector is used to periodically collect these dead nodes. During the garbage collection marks are propagated back along the traceback by starting at the pointers contained in the active tokens. At the end of this process any node that is not marked is inaccessible from any active hypothesis and can be removed and the memory reclaimed. When the decoder is operating in live mode at this point we also look for a common prefix in the traceback that can be output immediately. In previous work we investigated a multi-threaded decoder implementation. We recently re-visited the multi-threading issue and evaluated more recent libraries in particular Intel's Thread Building Blocks (TBB)[1]. The use of the data parallel library massively simplified adding multiple processor support by allowing the decoding loops to be divided across cores. Unfortunately we were only able to achieve speed-ups similar to those reported earlier[9].

IV. Output Formatting

The decoder can output recognition results in single best path and phone lattice output formats. The phone lattices are represented as a WFST and this means we can apply standard WFST operations to transform them into other representations such as word lattices, n-best lists[27] or confidence scores.

V. WFST Interface

The purpose of the WFST abstraction is to provide a consistent approach to various underlying WFST implementations including the disk based representations and the on-the-fly composition implementations. The design of the WFST interface follows the design described in [22].

[1]http://www.threadingbuildingblocks.org/

A. Generalized On-the-fly Algorithms

Lazy or on-the-fly composition techniques in speech recognition [11] are motivated by the need to reduce memory consumption both during network construction and decoding phases. The on-the-fly techniques also allow the component knowledge sources to be changed and modified more easily. The memory reduction occurs because the size of the final composed machine is often greater than sum of the component transducers. The T^3 decoder can directly use several component transducers thus making on-line switching and modifications much less costly.

However, performing on-the-fly composition during decoding causes performance reduction due to a combination of the overhead of the actual combination and the omission of optimizations that would normally be applied to the fully composed WFST.

In the T^3 decoder we have developed a generalized fast on-the-fly scheme that differs from previous approaches in several key areas that will be explained in the following sections. The technique is fully implemented within the WFST abstraction removing any requirement for *ad-hoc* decoder modifications. Our approach is a generalized version of the operations proposed by Caseiro and are extended to make use of a special composition filter that circumvents the restrictions Caseiro placed on topology of the individual WFSTs [25].

B. Weighted Composition

Given the WFST L which will map from x to y with weight w_l and the WFST R which maps from y to z with the weighting w_r, the composition WFST $L \circ R$ will map from x to z with $w_l \times w_r$ in the tropical semiring. Each state in the composed transducer $L \circ R$ is represented as a pair (q_l, q_r) that corresponds to the states q_l and q_r from the transducers L and R. The arcs of each state (q_l, q_r) in $L \circ R$ are computed using the following:

1) When the left transducer has an epsilon output label $(o_l = \epsilon)$ create the arc $((q_l, q_r), i_l, \epsilon, w_l, (q_l^{'}, q_r))$.
2) When the right transducer has an epsilon input label $(i_r = \epsilon)$ create an arc $((q_l, q_r), \epsilon, o_r, w_r, (q_l, q_r^{'}))$.
3) When the symbols match, that is $o_l = i_r$ create an arc $((q_l, q_r), i_l, o_r, w_l \otimes w_r, (q_l^{'}, q_r^{'}))$. Epsilon labels are also treated the same as regular labels and in the case when $o_l = \epsilon$ and $i_r = \epsilon$ we also generate an arc.

Here, the tuple $(q, i, o, w, q^{'})$ denotes an arc with the source state q, input symbol i, output symbol o, weight w and a destination state $q^{'}$.

VI. Optimization in On-the-fly Composition

Caseiro[8] proposed the following on-line algorithms that perform optimizations during on-the-fly composition: *dead-end state avoidance*, *dynamic pushing* and *state sharing*, however these approaches placed topological restrictions on the input WFSTs.

For the T^3 decoder we have developed generalized versions of these optimization operations and the key improvement is the ability to accept transducers of arbitrary structure.

In the following section we describe the *dead-end state avoidance* and *dynamic pushing* operations that were originally proposed by Caseiro then describe our extended implementations.

A. Dead-end State Avoidance

Any state which is not part of a path to valid final state is called a *non-coaccessible* or *dead-end* state. These useless states can be generated during the composition procedure because of delays in symbols matching or the presence of epsilon arcs. When performing the composition off-line a post-processing step is often employed to remove the dead-end states and ensure the final WFST is *trim*. For practical ASR it is essential to remove such states because they waste resources and negatively effect the performance of the decoder. However, when using on-the-fly composition removing such states is much more difficult.

In [8] Caseiro proposed a specialized on-the-fly composition procedure that would avoid the generation of dead-end states. There are two extensions in the method, the first modification is a pre-processing where for every state in the lexicon transducer L an *anticipated label set* is constructed. Each of these anticipated label sets will contain all of the outputs labels that can be reached from the hosting state. The second modification occurs during on-line composition, given a pair of states (q_l, q_r) we take the intersection of the anticipated label set in state q_l with the input label set from q_r. In the case when the intersection is empty there are no valid future matches and no arcs will be generated.

However, when there are arcs with epsilon input labels at state q_r the application of the above procedure can often lead to difficulties . If the state q_r has only epsilon input labels, the intersection becomes empty causing no arcs to be generated and an incorrect WFST. Caseiro's solution is to only allow ϵ input transitions in R when q_l is the initial state of L. Otherwise, use composition rules 2 and 3 and to prevent the excessive generation of dead-end states restrict L in the following ways:

- The transducer must loop through the initial state.
- Each path between the initial and final state must output only one label.

We proposed[28], [29] a generalized extension which imposes no topological restrictions by utilizing a composition filter to remove redundant paths from the composed WFST.

In filter composition the ϵ outputs in L are substituted with ϵ_2 symbols and ϵ inputs in R are substituted with ϵ_1. A self loop with output ϵ_1 is added to every state in L and a self loop with input ϵ_2 is added to every state in R. These transducers are denoted $L^{'}$ and $R^{'}$ respectively. Performing the composition $L^{'} \circ F \circ R^{'}$, where F is the composition filter will give a WFST with the redundant paths removed. With the introduction of a filter the following three way composition of L, F and R gives states identified with the following tuple (q_l, q_f, q_r).

The additional parameter q_f is one of the three filter states which we can enter as follows; State zero is composition rule

3 - generation by symbol matching. State one is generation by ϵ input and state two is generation by ϵ output. The filter does not permit a transition between states one and two, which means the composition WFST will not have the paths which contain an ϵ output (input) transition followed by an ϵ input (output) transition before a transition by symbol matching.

The proposed method exploits these transition restrictions to perform dead-end state avoidance and is implemented by application of the following rules given the state triplet (q_l, q_f, q_r) .

1) If $q_f = 0$ and q_r has no ϵ input transition, then perform standard dead-end checking with intersection of q_l and q_r.
2) If $q_f = 1$ and q_l has only ϵ output transitions, do not expand.
3) If $q_f = 2$ then perform standard dead-end checking with intersection of q_l and q_r.

B. Dynamic Pushing

The pushing operation is used to move either weights or labels closer to either the initial or final states of WFST. Pushing weights closer to the initial state is informally a similar idea to language model look-ahead [30], during the decoding process the weights are encountered earlier and this influences the pruning in a beneficial way that can help to improve recognition speed.

Caseiro also developed an on-the-fly approximation to weight pushing known as *dynamic pushing* [8]. To avoid the difficulty in calculating best path scores and re-weighting, he instead used the following look-ahead approximation. When the arcs are generated by matching ϵ outputs, select the smallest arc weight from q_r whose input is also contained in the anticipated label set of q_l, or for all the other matching cases do not set a lookahead score. Epsilon edges can lead to conflicts when the two heuristics attempt to set two different scores in a state, however, the use of the composition filter can be again used to rectify the situation. Under filter composition the matching rules will lead to the generation of different composition states depending on the matching rules and therefore no collisions in the lookahead heuristics will occur.

VII. Acoustic Scoring

A. Improvements in Acoustic Scoring

The acoustic scoring interface has several invocation points that are activated during the decoding cycle. A method is invoked to request a state score, after which the underlying implementation can either compute the score on-demand or return a pre-cached value. Signals are given at the start and end of frame which will indicate the acoustic model states that will be active for the next arc propagation. These signals allow for blocks of scores to be computed and cached inside the acoustic scoring module, furthermore the ability to *peak* the feature stream is also available to permit windowing and lookahead caching of the acoustic scores.

In T^3 we introduced the technique of using the graphics card for acoustic scoring[9], [10]. Current GPUs are programmable processors that are massively parallel. Recently a field has emerged known as General Purpose GPU (GPGPU) which aims to make use of the high floating point performance offered by GPUs for non-graphics application. In the field of GPGPU many computational intensive tasks have been moved off the CPU to GPU and often very large speed improvements are reported. Our GPGPU scheme is programmed through CUDA for use with NVIDIA GPUs.

As part of the development process and to provide a selection of techniques to compare the GPU we also implemented several CPU baselines. Before discussing the GPU in more technique we will give a brief overview of the CPU scoring techniques available and how they perform in combination.

The acoustic models in T^3 are weighted Gaussian mixture models (GMMs) of the form:

$$\log p(\mathbf{x}) = \log\left(\sum_i w_i p_i(\mathbf{x} \mid \theta_i)\right) \tag{2}$$

Where, $\mathbf{x}$ denotes the acoustic feature vector, w_i the prior probabilities and each diagonal covariance Gaussian density p_i is parameterized by θ_i and calculated according to:

$$\frac{w_i}{2\pi^{\frac{J}{2}}\left(\prod_j \sigma_{ij}^2\right)^{\frac{1}{2}}} \exp\left(-\sum_j \left(\frac{x_{ij} - \mu_{ij}}{\sigma_{ij}}\right)^2\right) \tag{3}$$

Where w_i is the weight, μ_{ij} and σ_{ij}^2 are the mean and variance in the j^{th} dimension. When scoring with the CPU the logsum in equation 2 is extremely expensive and the following *logmax* substitution is used instead:

$$\log p(\mathbf{x}) \approx \max_i \log \mathcal{N}(\mathbf{x} \mid \theta_i) \tag{4}$$

This alone has yielded good speed-ups with a very small reduction in recognition performance.

To obtain further speed-ups we use the SSE instructions which allow four floating operations to be computed in parallel. The acoustic score cache uses a likelihood batching scheme that compute the scores for an additional block of K future frames as described in [31]. This frame caching helps to make better use of memory bandwidth which is the main computation bottleneck. T^3 can also distribute the acoustic scoring across multiple cores, however we have only observed good speedups when using the full logsum, because under the logmax approximation memory is the bottleneck and we have not observed any substantial speedups when using more than one core. We also have implemented dimensional pruning[19] and noticed in practice either dimensional pruning or SSE leads to similar gains, we opt for SSE as it gives the slightly larger speed-up of the two.

By far our most successful speedup technique has been to use the GPU to perform the acoustic scoring whilst the CPU performs all the other computations. The next section contains an explanation of the technique and the methods we use to ensure the GPU is kept working as much as possible during decoding.

1) Approach to GPU Based Likelihood Computations: The GPU approach is a massively parallel approach that computes every Gaussian in parallel using a highly efficient matrix-matrix multiplication routine [32]. Unlike the CPU scheme a full logsum is used because the GPU can perform the logarithm and exponential much faster and the computation can be further accelerated by using the large scale parallelism of the GPU.

To compute all the acoustic scores together the following matrix multiplication is performed $\mathbf{y} = \mathbf{AW}$ [31], where $\mathbf{A}$ has a row corresponding to each of the Gaussian components in every GMM, and W is a window of feature vectors. Each row in $\mathbf{A}$ is derived from equation 3 using:

$$\left\{ K_i, \frac{\mu_{i1}}{\sigma_{i1}^2}, \ldots, \frac{\mu_{ij}}{\sigma_{ij}^2}, -\frac{1}{2\sigma_{i1}^2}, \ldots, -\frac{1}{2\sigma_{ij}^2} \right\} \tag{5}$$

K_i is computed using:

$$\log w_i - \frac{J}{2}\log 2\pi - \frac{1}{2}\sum_j \log \sigma_{ij}^2 - \frac{1}{2}\sum_j \frac{\mu_{ij}^2}{\sigma_{ij}^2} \tag{6}$$

Each column in $\mathbf{W}$ is an an acoustic feature vector $\mathbf{x}$ expanding to:

$$\mathbf{x}' = \{1, \mathbf{x}_1, \ldots, \mathbf{x}_J, \mathbf{x}_1^2, \ldots, \mathbf{x}_J^2\} \tag{7}$$

The result of the multiplication is a matrix $\mathbf{y}$ which contains the log weighted scores for every Gaussian component for every state mixture model across the windows of features vectors.

The next step is to compute the logsum of equation 2, in this step we launch a large amount of threads to calculate all of the logsums in parallel by using a reduction pattern. This approach ensures the memory bandwidth is used effectively on the GPU.

There is no need for the on-demand interface discussed previously because we have all of the acoustic scores for the window in a buffer. All of the work and communication with the GPU is hidden in to the function which is called at the beginning of frame search and all of the expansion and communications is implemented here in a manner that does not require other changes to the main decoding loop. Because the acoustic model can peak into the stream of feature vectors we can in fact make a further improvement and perform work concurrently. The GPU's *asynchronous* launches make it possible for the next window of samples to be batched off to the GPU whilst the CPU decoder searches the current window of scores. Under the hood the acoustic model implementation launches the transfers and computations. In the case where the searching completes before the next window of scores has been calculated the acoustic model can block the main decoding loop until it is safe to continue.

B. Miscellaneous Extensions

The GPU method can be extended in various ways, one technique is the ability to partition the acoustic model and distribute them across multiple GPUs. In this case we use a set of worker threads to co-ordinate the multiple graphics cards as opposed to the asynchronous technique described in the previous section. For our current system and models the multi-GPU technique does not lead to any performance improvements as a single GPU is adequate, however, the technique has been successively used with acoustic models that are too large to fit into the memory of an individual graphics card. We also have the ability to essentially perform the opposite and use a single GPU to perform scoring on multiple set of acoustic models in parallel for use in multi-stream scenarios.

VIII. Robust Voice Activity Detector

We have recently developed a novel Voice Activity Detector (VAD) that integrates into both the front-end and the acoustic scoring part of the decoding engine. The VAD technique illustrates the flexibility of T^3 and how the multiple extension points are used for developing and implementing new ideas and techniques.

The basic idea is when a frame acoustic score is requested for a silence type model a confidence measure of non-speech is added. When the state belongs to a phonetic model a confidence measure of speech is added. By using the speech and non-speech confidence score as a continuous value rather than a hard binary decision inside the decoder it should be possible to perform a better discrimination of speech and silences.

The first step is to add a new filter type to the front-end which contains the speech and non-speech Gaussian mixture models that are used to compute the likelihoods $p(X^i|H_1)$ and $p(X^i|H_0)$ for the i^{th} frame, where H_1 is the hypothesis of speech and H_0 is the hypothesis of non-speech. If the scores strongly indicate the frame contains no speech, these non-speech frames can be discarded early in the recognition pipeline, and thus the computational cost is reduced as not all of the input needs to be processed by the decoder . Otherwise, these likelihoods are then used to calculate the following confidence measures:

$$C_{H_1}^i = \frac{p(X^i|H_1)}{p(X^i|H_1) + p(X^i|H_0)} \tag{8}$$

$$C_{H_0}^i = \frac{p(X^i|H_0)}{p(X^i|H_1) + p(X^i|H_0)} \tag{9}$$

Where $C_{H_1}^i$ is the confidence measure of the frame containing speech and $C_{H_0}^i$ represents the confidence measure of the frame not containing speech. The denominator terms ensure the confidence measures exist in the range 0 to 1.

The confidence measures are added into the data packet containing the feature vector payload and passed thought the recognition engine without influencing any other components of the system. An extended acoustic model is plugged into the engine that will use the confidence measures to bias the acoustic model as follows. In the case where the recognition hypothesis in the token belongs to a phone model the acoustic

model score is biased using:

$$\log\ \hat{p}_{am}(X^i|\theta) = \log\ p_{am}(X^i|\theta) + \alpha\ \log\ \bar{C}^i_{H_1} \quad (10)$$

$$\bar{C}^i_{H_1} = \frac{\sum_{i-n}^{i+n} C^i_{H_1}}{2n+1} \quad (11)$$

Otherwise the hypothesis must belong to a silence model and the acoustic model score is biased using:

$$\log\ \hat{p}_{am}(X^i|\theta) = \log\ p_{am}(X^i|\theta) + \alpha\ \log\ \bar{C}^i_{H_0} \quad (12)$$

$$\bar{C}^i_{H_0} = \frac{\sum_{i-n}^{i+n} C^i_{H_0}}{2n+1} \quad (13)$$

Where X^i is the i^{th} feature vector, θ is hypothesis, $p_{am}(X^i|\theta)$ is the acoustic model score, α is a scaling factor and n is a smoothing parameter for computing $C^i_{H_1}$ and $C^i_{H_0}$ over a window.

When the α scaling parameter is set to zero the biased acoustic scores $\log\ \hat{p}_{am}(X^i|\theta)$ are equivalent to the standard non-biased scores $\log\ p_{am}(X^i|\theta)$.

In the interval where $C^i_{H_1}$ becomes one, the acoustic model scores in the word models becomes $\log\ p_{am}(X^i|\theta)$ and the score from the silence model will tend to $-\infty$, and therefore it is only possible to recognize a word in this interval. Similarly, when $C^i_{H_0}$ is one it will only become possible to allow silence hypotheses in this interval. Under these perfect conditions the proposed method will have no affect on the acoustic scores.

Part of the following section contains a summary of some recent evaluations that illustrate the performance of our VAD technique.

IX. Experiments

In this section we present several sets of evaluations that illustrate the performance of the techniques previously described.

A. GPU Evaluations

This section provides a recent evaluation on the Corpus of Spontaneous Japanese (CSJ) using the following training and testing protocols[20].

The speech waveforms were first converted to sequences of 39 dimensional feature vectors with 10 ms frame rate and 25 ms window size. Each feature vector was composed of 12 MFCCs with deltas and delta-deltas, augmented with log energy, log delta and log delta-delta energy terms.

The acoustic models were three-state left-to-right HMM tri-phone models and the complexity of GMM state models contained power-of-two sizes from four to 512 inclusive. Regular EM training with splitting was performed utilizing the data from 967 lectures.

The Kneser-Ney smoothed tri-gram language model G was constructed using the AT&T GRM toolkit [3]. The $C \circ L \circ G$ search network was composed and optimized using the `dmake` tool from the AT&T DCD toolkit [2]. The final search network contained approximately 3M states and 6M arcs.

The test set used for the evaluation was composed of 2338 utterances which spanned 10 lectures. This yielded a total of 116 minutes of speech. The experiments were conducted on a 2.4 GHz Intel Core2 machines with 2GB of memory.

In this evaluation we used an NVIDIA 8800GTX card which is equipped with a 128 core G80 GPU. The G80 is implemented as a set of eight multiprocessors, where each multiprocessor is a Single Instruction Multiple Data (SIMD) processor containing 16 cores. Each of the multiprocessor cores execute the same instruction stream in parallel operating on different portions of data.

The results in Figure 2 show several important results. Firstly in terms of accuracy even though we are only using standard EM training we are able to achieve very good recognition accuracy. The figure clearly illustrates a key benefit of using the GPU for acoustic scoring. When the search parameters are fixed there is virtually no increase in the Real Time Factor (RTF) as the GMM complexity is increased.

When compared to the CPU baseline using a lookahead cache of 10 frames the GPU accelerated scheme is on average over two times faster. For these best speed-up case approximately half of the CPU time was spent on the acoustic scoring. By using the GPU acceleration we are essentially removing all of this cost off the CPU. The acoustic likelihood computations which are traditionally the most expensive part of search, are now just a few percent of the runtime in T^3. A drop of accuracy is observed when using the 512 this is due to insufficient training data to robustly train these large acoustic models.

B. On-the-fly Evaluations

In these evaluation we again used CSJ for training and testing to explore various aspects of our proposed on-the-fly algorithms. As in the previous section 39 dimensional acoustic feature vectors were used in combination with acoustic models that had 32 Gaussians per mixture . The language model was Katz smoothed [17] back-off tri-gram with a vocabulary of 65k words trained on the 2,682 lectures of data and pruned to give approximately 199,636 states and 1,422,453 arcs.

The recognition cascade $(H \circ C) \circ L \circ G$ consisted of three WFSTs $(H \circ C)$, L and G combined using our on-the-fly composition with optimization. To understand the benefits of our approach we compare the recognition cascade to a fully pre-composed and optimized cascade $(H \circ C \circ L \circ G)$. We also used another baseline cascade that used Caseiro method in which the L and G WFSTs were composed with on-the-fly optimizations as describe in [8] and $(H \circ C)$ was composed without any optimizations.

For efficiency reasons inside the on-the-fly WFST implementation a cache technique was used to hold the arc information once a state has been composed on-the-fly. Before decoding each new utterance the cache was cleared.

For each of the composition methods the total parameter counts for the search space is given in Table 1. The values reveal that all of the techniques generate a composition WFST $L \circ G$ of approximately the same size. The slight size increase

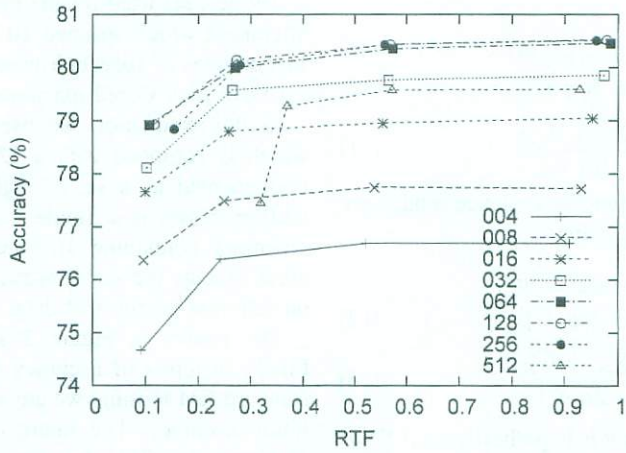

Fig. 2. Accuracy vs Real Time Factor (RTF) when using GPU acoustic scoring on the CSJ task. The acoustic models had between four to 512 Gaussians per mixture where each model complexity is shown as a separate curve.

TABLE I
THE WFST PARAMETER COUNTS FOR THE VARIOUS COMPOSITION METHODS.

Network	Method	Ave. Memory	#states	#arcs
$L \circ G$	Static	N/A	1,031,229	2,207,982
$L \circ G$	Caseiro	N/A	1,040,267	2,251,670
$L \circ G$	Proposed	N/A	1,198,351	2,409,754
$(H \circ C \circ L \circ G)$	Static	152MB	1,091,075	2,437,298
$(H \circ C) \circ L \circ G$	Caseiro	133MB	25,357,146	34,054,721
$(H \circ C) \circ L \circ G$	Proposed	136MB	5,505,110	9,433,757

in our method over the Caseiro method is because of extra states which the composition filter inserts.

In the case of $(H \circ C) \circ L \circ G$ WFST we are able to produce a network with substantially fewer arcs and states when compared to the Caseiro approach, which is because of the additional optimizations applied when $(H \circ C)$ is composed with $L \circ G$. In the Caseiro approach we use a standard composition at this stage, which causes many more dead-end states to be created. The standard static approach achieves a much smaller search space. This is because the final WFST has no dead-end states, more optimization can be applied and further size reductions are achieved by factoring the phone arcs in longer chains.

Figure 3 shows the RTF vs accuracy for the various composition approaches. In this figure the static network is fully pre-composed and optimized, whilst the other techniques contain two on-the-fly operations. The result shows that the proposed method decodes slightly faster than the standard Caseiro method.

C. VAD Evaluations

We evaluated our VAD approach using the Drivers Japanese Speech Corpus in a Car Environment (DJSC) corpus [12]. This is a hands-free command and control task composed of utterances recorded in a car driving on a motorway. The test set consists of 40 speakers equally split between male and female speakers. Each participant provided 41 commands in an utterance continuously that would operate navigation whilst driving. The commands within each utterance are separated by one to two seconds non-speech regions which capture the background noise conditions. The recordings were performed at 16 kHz using a microphone mounted in the position of the navigation device. The acoustic models were trained on 52 hours of speech data from the Japanese Newspaper Article Sentences (JNAS [15]) corpus. The training material is gender balanced containing 130 male speakers giving 25 hours of speech and 130 female speakers providing another 27 hours of speech.

From the processed data the acoustic models were EM trained and this process yielded a set of three states left-to-right tri-phone HMM with 2000 states. Each state output density was a 16 component GMM with diagonal covariance. In the evaluation the training and testing data were processed as follows; The raw speech waveforms were converted to a sequence of 38 dimensional feature vectors with 10 ms frame rate and 25 ms windows size. Each feature vector was composed of 12 MFCCs with deltas and delta-deltas, augmented with log delta and delta-delta energy terms.

The language model was a network grammar and the vocabulary size was 83 words to cover all of the commands. The network had a path which corresponded to each of the valid commands that looped through the initial state to allow continuous recognition of the utterance stream.

The GMMs for the VAD each had four Gaussian components. The speech GMM was trained using the data from 967 CSJ lectures and the non-speech GMM was trained with data from car noise from Japan Electronic Industry Development Association (JEIDA).

Figure 4 show the recognition accuracy when using the proposed VAD methods and two baselines, where:

- *baseline* represents the result without any VAD.
- *ZCR* is the result when using zero crossing rate[5] and

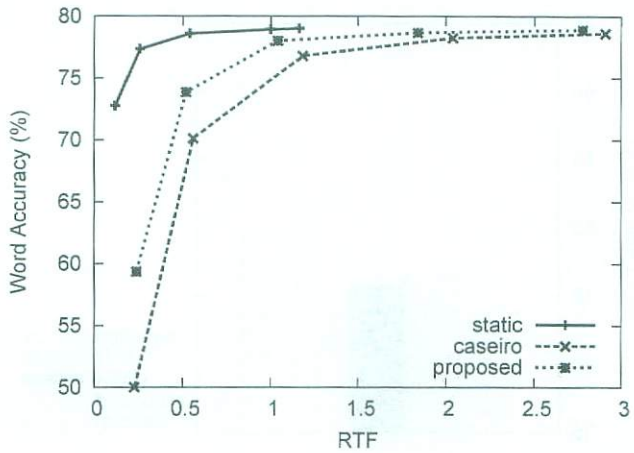

Fig. 3. Accuracy vs RTF when using static and on-the-fly composition schemes.

energy for VAD (front-end VAD).

- *GMM* is the result using the GMM based likelihood ratio detector (front-end VAD).
- *proposed* is the result of our proposed VAD.
- *manual* corresponds to the result when using the corpus labels to remove non-speech region.

The parameters of each of the VADs were optimized manually to the following:

- In the ZCR system the power threshold was set to 0, the threshold of zero crossing rate was set to 10 and the frame length was 25ms.
- In GMM VAD the likelihood ratio threshold was -5.
- In the proposed method the scaling factor α was set to 10 and smoothing parameter n was set to 15.
- The thresholds and smoothing parameters of the VADs where optimized and fixed for the whole target test set.

The scores $C^i_{H_1}$ and $C^i_{H_0}$ were calculated using the same GMMs as the GMM based likelihood VAD.

The results show that without any VAD the word recognition accuracy was 43.1%. The ZCR method achieved 46.5% and GMM method achieved 45.8% word accuracy. These corresponded to an absolute word improvement of 3.4% and 2.7% respectively. The proposed technique achieved a 53.1% word accuracy and this corresponded to an absolute 10.0% improvement over the non-VAD baseline; this was the highest word accuracy we obtained in the evaluations and this result shows the effectiveness of our method. Using the labels from the corpus 60.4% word accuracy was achieved and this indicates that there is still room to improve the proposed method.

X. Conclusions

In this paper we have given an overview of the T^3 WFST decoding engine and described the main features and components. We have provided a more in-depth description of the recent developments including GPU based acoustic scoring, generalized on-the-fly composition and optimization and a decoder integrated VAD scheme.

Our evaluations show by using the GPU we can achieve good accuracy on the CSJ at a very high recognition speed. The on-the-fly results show that our proposed method can outperform the operations proposed by Caseiro in both WFST size and accuracy given the same parameters. More importantly the proposed method can accept WFSTs of un-restricted topology. The VAD approach has been shown to achieve a large improvement in word recognition under real noisy conditions.

In future work we not only want to continue to add new features to the engine but also make improvements to the current features, such as further improving the the on-the-fly operations. In addition we are planning to add more functionality to the engine and are also interested in porting it to more platforms. We are currently looking at more applications of the engine such as integrating machine translation.

Acknowledgment

This work was supported by the Japanese government METI Project "Development of Fundamental Speech Recognition Technology". The authors wish to thank Takahiro Shinozaki for providing the acoustic models used in the GPU evaluations.

References

[1] C. Allauzen, M. Mohri, and M. Riley. Statistical modeling for unit selection in speech synthesis. In *Proc. of 42st Annual Meeting of the Assocication for Computational Linguistics*, pages 55–62, 2004.

[2] C. Allauzen, M. Mohri, M. Riley, and B. Roark. A generalized construction of integrated speech recognition transducers. In *Proc ICASSP*, 2004.

[3] C. Allauzen, M Mohri, and B. Roark. Generalized algorithms for constructing statistical language models. In *Proc. of 41st Annual Meeting of the Assocication for Computational Linguistics*, pages 40–47, 2003.

[4] 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.

[5] A. Benyassine, E. Shlomot, H. Y. Su, D. Massaloux, C. Lamblin, and J. P Petit. ITU-T recommendation G.729 annex B: a silence compression scheme for use with G.729 optimized for V.70 digital simultaneous voice and data applications. *IEEE Communications Magazine*, 35(9):64–73, 1977.

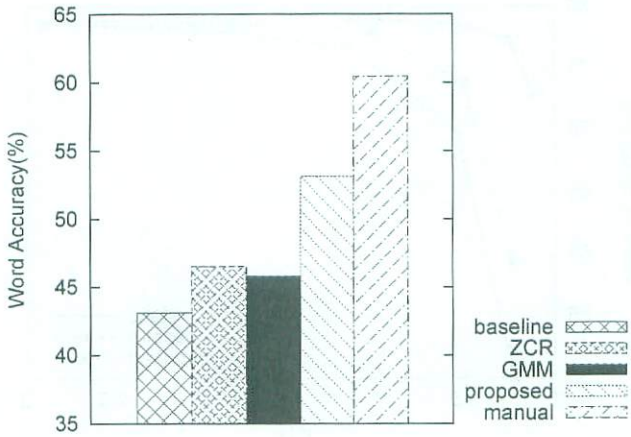

Fig. 4. The effect of VAD on recognition accuracy.

[6] I Bulyko and M Ostendorf. Unit selection for speech synthesis using splicing costs with weighted finite state transducers. In *Proc. EUROSPEECH*, pages 987–990, 2001.

[7] 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.

[8] D. A. Caseiro and I. Trancoso. A specialized on-the-fly algorithm for lexicon and language model composition. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(4):1281–1291, 2006.

[9] P. R. Dixon, D. A. Caseiro, T. Oonishi, and S. Furui. The Titech large vocabulary WFST speech recognition system. In *Proc. ASRU*, pages 1301–1304, 2007.

[10] P. R. Dixon, T. Oonishi, and S. Furui. Fast acoustic computations using graphics processors. In *Proc. ICASPP*, pages 4321–4324, 2009.

[11] Hans. J. G. A Dolfing and I.Lee Hetherington. Incremental language models for speech recognition using finite-state transducers. *Proc. ASRU*, pages 194–197, 2001.

[12] K. Hiraki, T. Shinozaki, K. Iwano, K. Shinoda A. Betkowska, and S. Furui. Initial evaluation of the driver's Japanese speech corpus in a car environment. In *IEICE Technical Reports, Asian Workshop on speech science and Technology*, pages 93–98, 2008.

[13] C. Hori, K. Ohtake, T. Misu, H. Kashioka, and S. Nakamura. A statistical approach to expandable spoken dialog systems using WFSTs. In *Universal Communication, 2008. ISUC '08. Second International Symposium on*, pages 24–27, 2008.

[14] T. Hori and A. Nakamura. Generalized fast on-the-fly composition algorithm for WFST-based speech recognition. In *Proc. INTERSPEECH*, pages 847–850, 2005.

[15] K. Itou, M. Yamamoto, K. Takeda, T. Takezawa, T. Matsuoka, T. Kobayashi, K. Shikano, and S. Itahashi. JNAS: Japanese speech corpus for large vocabulary continuous speech recognition research. In *J.Acoust. Soc. Jpn.(E)*, pages 199–206, 1999.

[16] S. Kanthak, H. Ney, M. Riley, and M. Mohri. A comparison of two LVR search optimization techniques. In *Proc. ICSLP*, pages 1309–1312, 2002.

[17] S. M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401, 1987.

[18] P Lamere, P Kwok, W Walker, E. Gouvea, R Singh, B Raj, and P Wolf. Design of the CMU sphinx-4 decoder. In *Proc. EUROSPEECH*, pages 1181–1184, 2003.

[19] A Lee, T Kawahara, and K Shikano. Julius — an open source real-time large vocabulary recognition engine. In *Proc. EUROSPEECH*, pages 1691–1694, 2001.

[20] K. Maekawa. Corpus of spontaneous Japanese: Its design and evaluation. In *Proc. ISCA and IEEE Workshop Spontaneous Speech Processing and Recognition*, pages 7–12, 2003.

[21] E. Matusov, S. Kanthak, and H. Ney. Integrating speech recognition and machine translation: Where do we stand? In *Proc. ICASSP*, pages 1217–1220, 2006.

[22] F. Mohri, M .and Pereira, O. Pereira, and M. Riley. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231:17–32, 2000.

[23] M. Mohri. Weighted automata algorithms. *Handbook of weighted automata*, (to appear) 2009.

[24] M Mohri, F. Pereira, O. Pereira, and M. Riley. A rational design for a weighted finite-state transducer library. In *Lecture Notes in Computer Science*, pages 144–158, 1998.

[25] M. Mohri, F. Pereira, and M. Riley. Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, 16(1):69–88, 2002.

[26] M. Mohri, F. C. N Pereira, and M. Riley. Speech recognition with weighted finite-state transducers. *Handbook on Speech Processing and Speech Communication*, 2008.

[27] M. Mohri and M. Riley. An efficient algorithm for the n-best-strings problem. In *Proc. ICSLP 02*, 2002.

[28] T. Oonishi, P. R. Dixon, K. Iwano, and S. Furui. Implementation and evaluation of fast on-the-fly WFST composition algorithms. In *Proc. INTERSPEECH*, pages 2110–2113, 2008.

[29] T. Oonishi, P. R. Dixon, K. Iwano, and S. Furui. Generalization of specialized on-the-fly composition. In *Proc. ICASPP*, pages 4317–4320, 2009.

[30] S. Ortmanns, H. Ney, and A. Eiden. Language-model look-ahead for large vocabulary speech recognition. *Proc. Spoken Language Processing*, pages 2095–2098, 1996.

[31] M. Saraclar, M. Riley, E. Bocchieri, and V. Goffin. Towards automatic closed captioning : Low latency real time broadcast news transcription. In *Proc. ICSLP*, pages 1741–1744, 2002.

[32] V Volkov and J W. Demmel. Benchmarking GPUs to tune dense linear algebra. In *Proc. ACM/IEEE conference on Supercomputing*, pages 1–11, 2008.

[33] S. Young, G. Evermann, D. Kershaw, G. Moore, J. Odell, D. Ollason, V. Valtchev, and P. Woodland. *The HTK Book (for HTK Version 3.2)*. Cambridge University Engineering Department, 2002.

[34] S.J. Young, N.H. Russell, and J.H.S. Thornton. Token passing: A simple conceptual model for connected speech recognition systems. Technical report, Cambridge University Engineering Department, 1989.