Available online at www.sciencedirect.com

ScienceDirect

Computer Speech and Language 23 (2009) 510–526

COMPUTER SPEECH AND LANGUAGE

www.elsevier.com/locate/csl

Harnessing graphics processors for the fast computation of acoustic likelihoods in speech recognition

Paul R. Dixon [*], Tasuku Oonishi, Sadaoki Furui

Department of Computer Science, Tokyo Institute of Technology, 2-12-1, Ookayama, Meguro-ku, Tokyo 152-8552, Japan

Received 18 July 2008; received in revised form 19 March 2009; accepted 19 March 2009
Available online 31 March 2009

Abstract

In large vocabulary continuous speech recognition (LVCSR) the acoustic model computations often account for the largest processing overhead. Our weighted finite state transducer (WFST) based decoding engine can utilize a commodity graphics processing unit (GPU) to perform the acoustic computations to move this burden off the main processor. In this paper we describe our new GPU scheme that can achieve a very substantial improvement in recognition speed whilst incurring no reduction in recognition accuracy. We evaluate the GPU technique on a large vocabulary spontaneous speech recognition task using a set of acoustic models with varying complexity and the results consistently show by using the GPU it is possible to reduce the recognition time with largest improvements occurring in systems with large numbers of Gaussians. For the systems which achieve the best accuracy we obtained between 2.5 and 3 times speed-ups. The faster decoding times translate to reductions in space, power and hardware costs by only requiring standard hardware that is already widely installed.
© 2009 Elsevier Ltd. All rights reserved.

Keywords: LVCSR; GPGPU; Novel hardware for ASR; WFST

1. Introduction

The goal of this work was to substantially accelerate a state-of-the-art large vocabulary speech decoder (Dixon et al., 2007) by harnessing the massive computational power offered by modern graphics cards.

Modern graphics processor units (GPUs) have evolved into massively parallel processors with huge floating point throughput fed from specialized memory with phenomenal bandwidth. For example the GTX8800 hardware used in this work has performance figures of ~350 floating point operations per second (Flops) and ~85 GB/s of memory bandwidth (NVIDIA Corporation, 2007). In recent years a field of research known as general purpose GPU (GPGPU) has emerged where GPUs are used to perform general purpose computation and in many fields substantial speed improvements have been reported (Owens et al., 2005). GPU performance has

[*] Corresponding author. Tel./fax: +81 3 5734 3481.
E-mail addresses: dixonp@furui.cs.titech.ac.jp (P.R. Dixon), oonishi@furui.cs.titech.ac.jp (T. Oonishi), furui@furui.cs.titech.ac.jp (S. Furui).

0885-2308/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.csl.2009.03.005

been increasing 2–2.5 times per year, whereas central processing unit (CPU) performance doubles approximately every 18 months (Manocha, 2005; Owen et al., 2005). If we can move part or even all of the decoder to execute on the GPU then we can ride these faster moving performance trends.

Parallelization of decoder search algorithms may become necessary even if we wish to make full use of future CPU power. Recently CPU performance advances have become increasingly driven by placing more and more cores on each chip as opposed to increasing clock frequency. If parallelization is inevitable then why not consider the massive power offered by the GPU. Not only are huge performance improvements waiting to be unlocked, but the use of GPU may also bring significant savings in power, space and cost. If such benefits can be achieved then the GPU technique would not just be applicable for exploiting dormant GPUs in desktop machines or retro-fitting older single-core machines, but highly beneficial in server and cluster computing where power and cooling now dominate the costs. The recent availability of the NVIDIA *Tesla* series of hardware designed specifically for scientific purpose usage clearly show GPGPU is becoming an important resource that cannot be ignored. Furthermore, by freeing the main processor of the acoustic computations more of main CPU can be devoted to other parts of the search algorithm or additional tasks inside the recognition process. In this paper, we present our latest GPU implementation that performs much better by executing more work on GPU in a concurrent manner, performs better batching of work to GPU, and utilizes faster transfers. Not only does the technique achieve a significant reduction in decoding time but we have also eliminated slow-downs for small models and narrower search beams.

To illustrate the benefits of using the GPU we evaluate the approach and present several performance characteristics including speed and power consumption.

In Dixon et al. (2007), we proposed a GPGPU technique to compute the acoustic likelihoods and reported a 25% reduction in speech decoding time when compared to a single instruction multiple data (SIMD) CPU based approach. Even though we successfully showed that a GPU used as a co-processor could accelerate a speech decoder, it was obvious there was much more potential in the GPU and greater speed-ups could still be harvested. One small drawback with the technique was a slow-down that occurred when using narrow beams. For narrower beams we observed the combined cost of both executing and sending data to and from the GPU was greater than performing the necessary work on the CPU.

We have observed the Gaussian computations consuming a large amount of processing power of the total CPU time. There has been considerable previous research dedicated to reducing or speeding up this computation. Previous techniques include clustering, pruning and look-aheads using less complicated models (Aubert, 1989; Chan et al., 2004; Knill et al., 1996; Saon et al., 2003, 2005). However, with the approximation based techniques there can be a speed accuracy trade-off. It would be very desirable if the GPU approach could bring large speed gains without any reduction in accuracy. This is a realistic goal as speed gains have also been reported by utilizing the simpler vector SIMD hardware on most modern processors in the form of SIMD streaming extensions (SSE) instructions (Goffin et al., 2005).

1.1. Paper organization

The rest of the paper is structured as follows. Section 2 is a brief description of speech recognition specifically within the weighted finite state transducer (WFST) framework. This is followed by a section describing the operation of our large vocabulary decoder. In Section 4, we discuss the programming and hardware framework used in our work. Section 5 is an explanation of the GPGPU extensions that we have added to our decoder. Section 6 is a rich set of experiments designed to illustrate the speed increase from using the GPU accelerated decoder and the associated power characteristics. The paper finishes with a section detailing conclusions and future work.

2. Background

The basic speech recognition problem can be expressed as:

$$\widehat{\mathbf{W}} = \arg\max_{\mathbf{W}} \{P(\mathbf{X}|\mathbf{W})P(\mathbf{W})\} \tag{1}$$

The task of the decoder is to find the most likely word sequence $\widehat{\mathbf{W}}$, given an observed sequence of speech feature vectors $\mathbf{X}$. The language model probability is denoted by $P(\mathbf{W})$ and $P(\mathbf{X}|\mathbf{W})$ is the score contribution from the other knowledge sources such as the acoustic models.

Within recent years the WFST framework has become the standard approach for expressing and manipulating the knowledge sources used within speech recognition. Within the WFST paradigm all the knowledge sources in the search space are combined together to form a static search network (Mohri et al., 2002). The composition often happens off-line before decoding and there exist powerful operations to manipulate and optimize the search networks (Mohri et al., 2002). Within this framework the decoder becomes agnostic to various knowledge sources, thus changes made to the information sources before the network compilation stage should not require modifications to the decoder.

3. Decoder description

Our current implementation is a single-pass time-synchronous Viterbi beam search decoder using a token passing scheme (Young et al., 1989). The decoder has been designed specifically with speech tasks in mind. We are primarily considering a recognition cascade which performs a transduction from context dependent phone arcs to word sequences, that is $C \circ L \circ G$ (CLevel) where C is the context dependency, L is the lexicon, G is the language model and $\circ$ denotes the composition operator. During the search the decoder dynamically expands the hidden markov model (HMM) arcs into state sequences. It is also possible to decode state level networks by composing acoustic models H into the recognition cascade $H \circ C \circ L \circ G$ (HLevel), and then simulating the self-transitions via an appropriate arc definition that contains a single HMM state.

The decoder's search algorithm keeps a list of active search states and a list of active arcs. States and arcs are considered active when they contain tokens. The algorithm consists of creating a token for the empty hypotheses, associating it with the initial state of the search network and then iterating the three following steps for each parameter vector in the utterance.

(1) In the first step, expand active states, tokens are propagated from each state to the initial state of every arc leaving the state.
(2) In the second step, expand active arcs, the tokens held in each arc are advanced to the next frame. This step uses a specialized time-synchronous Viterbi algorithm optimized to the arc topology. When a token reaches the last state in the arc, it is propagated to the following search state, and activating it if necessary.
(3) Finally, in the third step, epsilon propagation, tokens are propagated across epsilon input edges.

The best solution is recovered as the best hypothesis contained in a final state after processing the last frame of the utterance. The word histories are maintained in a trace-back structure and each of the active tokens contains a reference to a node in this structure. As a consequence of the pruning many paths exist in this trace-back lattice which are no longer referenced by active tokens. To reduce excessive memory consumption a garbage collector is used to cull these useless paths. For performance reasons a *mark-sweep* collection scheme is used as opposed to reference counting. At fixed intervals the garbage collector operates by starting at each active token and propagating marks backwards along the nodes in the trace-back. 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.

Two pruning strategies are currently used: *Beam* pruning where at each frame, hypotheses are culled if they have a cost greater than the combined current best cost plus beam width. *Histogram* pruning caps the maximum active hypotheses allowed to a fixed band width.

Our decoding engine is not just the search component, but is a fully functioning standalone real-time speech recognition engine (Dixon et al., 2007):

- There is a highly configurable streaming front-end for performing feature extraction.
- Several implementations for memory reduction such as disk based data structures and on-the-fly (Hori and Nakamura, 2005; Caseiro and Trancoso, 2006; Oonishi et al., 2008) composition.

- In live operating mode there is support for continuous partial hypothesis output.
- Single best path and lattice output formats.

We were fortunate to begin developing the decoder at a time when the shift to multi-core and asymmetric architectures such as the cell broadband engine (Gschwind et al., 2006) was beginning to emerge. We knew at the early stage that implementing a decoder capable of exploiting such processors was essential. In Dixon et al. (2007), we showed a multi-threaded search implementation that could lead to an improvement in performance, and we first proposed the use of a GPU as co-processor for speeding up speech decoding. This paper continues the GPU work exclusively and we present a significantly more advanced technique that makes significant speed-ups for both large and small models for all search parameters.

4. CUDA and NVIDIA G80 series hardware

Recently, NVIDIA released the compute unified device architecture (CUDA) (NVIDIA Corporation, 2007). This hardware and software framework massively simplifies the development of GPGPU programs. In previous generations of graphics hardware the entire pipeline, memory layout and programming tools were designed to perform only graphics operations. Developing GPGPU applications required immense skill as a knowledge of both the target application and 3D graphics programming were needed. The operations in the non-graphics algorithms would have to be replaced with graphics equivalents such as geometry operations and data would have to be mapped into the graphics specific memory structures. For the CUDA generation of hardware NVIDIA introduced a unified core design and the cores were further generalized to allow non-graphics applications to target GPUs more easily. The accompanying application programming interface (API) provides a `C` like language which features a much less restrictive memory and input/output (I/O) model as well as support for single precision floating point operations.

In these evaluations, we used an NVIDIA 8800GTX card that is equipped with a 128 core GPU. The GPU is implemented as a set of eight multiprocessors, where each multiprocessor is a SIMD processor containing 16 cores. Each of the multiprocessor's cores executes the same instruction stream in parallel on different data. Thread creation on a GPU is very cheap and this is just one of the crucial factors that differentiates a GPU from a modern multi-core CPU. In fact to get maximum performance from a GPU it is necessary to launch massive amounts of threads (in the order of thousands for the current generation of hardware).

5. Acoustic computation using graphics hardware

The GPU excels at certain types of computation and therefore when selecting algorithms for the GPGPU the essential first step is to select routines that will benefit most from the GPU architecture. With much more of the silicon area dedicated to computation the GPU excels at arithmetic intense computations such as dense matrix multiplication. Conversely, due to the lack of branch prediction hardware it would not be reasonable to expect to see large performance gains on code containing a large amount of conditional logic.

The acoustic model scoring is an ideal candidate for computation on the GPU because not only does the computation of acoustic model scores often account for the largest amount of CPU time during decoding (Knill et al., 1996; Saraclar et al., 2002) but, the computation can be expressed as matrix multiplication (Saraclar et al., 2002) and this is precisely the data parallel computation task the GPU thrives on.

In our GPGPU scheme all of the parameters in the acoustic models are represented as a matrix **A** similar to (Saraclar et al., 2002), where each row is a representation of a log weighted Gaussian component. The ith component of J dimensional mixture model is expressed as a vector of length $2J+1$ according to:

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

where for each mixture model μ_{ij} and σ_{ij} are the mean and variance for ith Gaussian component in the jth dimension. The constant term K_i for the ith Gaussian component in a mixture model is calculated according to:

$$\log w_i - \frac{J}{2}\log 2\pi - \frac{1}{2}\sum_j \log \sigma_{ij}^2 - \frac{1}{2}\sum_j \frac{\mu_{ij}^2}{\sigma_{ij}^2} \quad (3)$$

Given an acoustic feature vector $\mathbf{x}$ the score of every Gaussian of every mixture can be calculated simultaneously by first expanding the feature vector to:

$$\mathbf{z}^T = \{1, \mathbf{x}_1, \ldots, \mathbf{x}_J, \mathbf{x}_1^2, \ldots, \mathbf{x}_J^2\} \quad (4)$$

and then performing the matrix vector multiplication $\mathbf{y} = \mathbf{Az}$. The result is a vector $\mathbf{y}$ containing log weighted scores of the feature vector for every Gaussian component for every mixture model.

The buses that connect the CPU (host) and GPU (device) together are clocked much slower than other system components and therefore a major bottleneck in the system is the interaction between the host and device. In this paper we present several techniques that address this communication bottleneck such as reducing the size or frequency of the transfers, or by performing tasks in parallel to hide the overhead.

Batching smaller transfers into larger transfers is one of the techniques used to reduce the communication bottleneck between the device and host. In our decoder this involves sending a window of acoustic frames as opposed to an individual frame to the GPU for score calculation. On the device we now receive a matrix $\mathbf{B}$ that contains an n long batch of feature vectors in expanded form:

$$\mathbf{B} = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n\} \quad (5)$$

The scores for every Gaussian in every mixture for the entire window are computed using the matrix multiplication $\mathbf{F} = \mathbf{AB}$, each column in $\mathbf{F}$ contains the log weighted scores for every Gaussian component of every mixture model for the corresponding feature vector in $\mathbf{B}$. Another potential advantage of the window approach is the matrix–matrix multiplication single general matrix multiply (SGEMM) operation like the other level 3 basic linear algebra subprograms (BLAS) operation are often optimized more greatly than the level 1 BLAS matrix vector operations. The windowing approach introduces another parameter for controlling the window length and this can crudely adjust the ratio between the device communication overhead and window latency when waiting for the GPU.

To reduce the size of the transfers back to the host, the GPU also performs the logsum part of the acoustic computation:

$$\log p(\mathbf{x}) = \log\left(\sum_j w_j \mathcal{N}(\mathbf{x}|\theta_j)\right) \quad (6)$$

Because we are dealing with the log weighted Gaussian scores $\log w_j \mathcal{N}(\mathbf{x}|\theta_j)$ the necessary precautions are taken to avoid numerical underflows when performing the summation. The device logsum is a highly parallel implementation that makes use of the GPU hardware `log` and `exp` operations. A matrix of state mixture scores $\mathbf{M}$ is computed in parallel from the matrix of Gaussian scores $\mathbf{F}$. This gives a window $\times$ mixture models block of mixture model scores and increasing the Gaussians per mixture will not increase the size of the transfers during decoding. The host acoustic score cache logic is now redundant as we have the acoustic scores for the window in a compact block of memory. This small contiguous representation may also bring performance increases by reducing the cache misses as the acoustic scores are accessed during search, in a similar manner as described in (Saraclar et al., 2002; Saon et al., 2003).

Performing work on the GPU asynchronously allows the CPU and GPU to operate in parallel and can further increase speed by hiding the GPU compute and transfer latencies. When realized in the decoder, the basic idea is to have the CPU searching on the current window of feature vectors and at the same time have the GPU compute the next window of acoustic scores. Later on in the section we describe our solution which avoids the need for extra worker threads by using the CUDA asynchronous interface.

Finally by utilizing page-locked memory for the host-device transfer buffers it may be possible to obtain a further small speed-up. If a host buffer is allocated using standard page-able memory every time a transfer is initiated the CUDA drivers will allocate a temporary page-locked buffer to hold a copy of the data whilst performing the transfers.

5.1. Decoder modification

The GPGPU implementation was integrated into the decoder currently under development at Tokyo Institute of Technology (Dixon et al., 2007). The decoding engine is a time-synchronous Viterbi beam system that operates on WFST (Mohri et al., 2002) search spaces. In the remainder of the section we discuss how our implementation executes the acoustic scoring on the GPU.

5.1.1. Initialization

The decoder first expands all of the acoustic models into matrix form according to Eq. (2), next allocates a buffer on the GPU and then transfers the model parameters to the GPU. In addition buffers are also allocated on the GPU to hold the window of feature vectors, Gaussian scores and mixture scores.

5.1.2. GPU interaction

In CUDA kernels are asynchronous (NVIDIA Corporation, 2007), that is, after invocation control will return immediately to the host. Memory transfers to and from the GPU can be either blocking and non-blocking. In our implementation to allow the GPU based acoustic scoring and the CPU search to operate concurrently we make use of the non-blocking memory transfers. All of the necessary GPU interactions are wrapped into two methods that are called in the main decoding loop. A compute window method instructs the GPU via a CUDA stream to perform memory transfers in addition to the matrix multiplication and the logsum. The use of the stream permits all of the methods to be called from a single host invocation point and returns control immediately to the host. The GPU will then sequentially execute each method in the stream and on completion will record a CUDA event. A wait method polls the event and blocks until it is set. Once the event has been set this indicates the compute method has completed and control can be safely returned to the host.

5.1.3. GPU compute window

Writes the following to a stream and returns control to host immediately.

(1) Record start event.
(2) Transfer feature vectors from host to device.
(3) Launch matrix multiplication kernel of the feature vector matrix with the parameter matrix storing the result in the Gaussian score buffer. The matrix multiplication utilizes the highly tuned routine described in Volkov and Demmel (2008).
(4) Launch logsum kernel to compute each of the mixture score from the Gaussian scores according to Eq. (6). The mixtures are uniformly assigned amongst the processors and the computation is launched as a block of threads.
(5) Transfer mixtures score from host to device.
(6) Record stop event.

5.1.4. GPU wait

Query the stop event and block until the compute window method completes.

5.1.5. Modified decoding loop

The main decoding loop is modified to perform transactions with the GPU before performing the main token passing search algorithm. Given an utterance of speech the main decoding loop becomes:

```
for Frame in utterance do
    if Host score cache is empty then
        Call GPU Wait to block for last compute to finish
        if Future frame left to score then
            Expand features according to Eq. (4)
            Call GPU Compute Window
        end if
    end if
    Perform main token propagation (as described in Section 3) using acoustic scores from the host cache.
end for
```

A special case occurs at the start of the search where the main token propagation must be blocked until the first block of acoustic scores is computed.

6. Experiments

In this section, we first describe the task, models and hardware setup. Next in this section, we present several evaluations that illustrate various aspects of the GPU decoder.[1] In these evaluations our primary interest is the speed of the decoder and the metric we will often use in comparisons is the real-time factor (RTF) calculated according to:

$$\mathrm{RTF} = \frac{\text{Time taken to decode speech}}{\text{Length of speech}} \tag{7}$$

The other metric we will frequently use is accuracy defined as:

$$\%\mathrm{Accuracy} = 100 \times \frac{\text{Ref labels} - \text{Substitutions} - \text{Deletions} - \text{Insertions}}{\text{Ref labels}} \tag{8}$$

When considering the speed change of the decoder after enabling the GPU we consider two metrics. The speed-up factor defined as:

$$\text{Speed-up} = \frac{\text{CPU Decoding Time}}{\text{GPU Decoding Time}} \tag{9}$$

And the percent reduction in decoding time (or RTF) is calculated according to:

$$\text{Percent Reduction} = \frac{\text{CPU Decoding Time-GPU Decoding Time}}{\text{CPU Decoding}} \times 100 \tag{10}$$

6.1. Experimental setup

6.1.1. Corpus

Our evaluations were carried out using the corpus of spontaneous Japanese (CSJ) which has recently become one of the standard datasets for building and evaluating state-of-the-art large vocabulary Japanese speech recognition systems (Kawahara et al., 2003; Maekawa, 2003). CSJ is a large spontaneous speech corpus consisting of approximately seven million words, the recordings are mainly of lectures, oral presentations or extemporaneous speeches (Kawahara et al., 2003).

6.1.2. Acoustic features

The original 16 kHz raw speech was first segmented into utterances, then converted to a standard 39 dimensional mel-frequency cepstral coefficients (MFCC) based parameterizations with a 10 ms frame rate and 25 ms window size. Each feature vector was composed of 12 MFCC values and an energy term all augmented with Δ and $\Delta\Delta$ values (Furui, 1986). Cepstral mean normalization was applied across the whole utterances and the energy term was removed to give the 38 dimensional feature representations.

[1] Even though the GPU is a co-processor we will sometimes refer to the decoder that uses the GPU for acoustic scoring as the GPU or GPGPU decoder.

6.1.3. Acoustic models

With the focus in this paper firmly set on accelerating the acoustic computation component of the decoder, a rich set of acoustic models of various complexities were built to allow us to investigate the speed characteristics of the GPU decoder. The topology of all the HMM acoustic models was a three state left-to-right tri-phone model. The training procedure was a standard EM approach with splitting using CSJ approximately 233 h of training data. The final sets of mixture models each contained 3000 states with power-of-two Gaussian components counts from 2 to 512 per mixture each with diagonal covariance. On the CSJ task we saw best accuracy for the 128 component mixtures and accuracy degradations for 256 and 512 component mixtures. We have presented the results for the larger models to illustrate the GPU approach has the ability to rapidly compute against large models and systems with large numbers of Gaussians with very little slow-down.

6.1.4. Language model

The language model was a back-off trigram using Katz smoothing with a vocabulary of 50 k words trained using CSJ training data of two million sentences (Kawahara et al., 2003).

6.1.5. Search network

The knowledge sources were used to create a $C \circ L \circ G$ recognition cascade containing ~3 M states and ~5.9 M arcs which gave a 138MB compiled search network, the phone arcs were dynamically expanded in the decoder.

6.1.6. Test set

For our evaluations CSJ *test-set 1* (Kawahara et al., 2003) was used. This test set comprises of 10 lectures totaling 116 min of speech and after segmentation yielded 2328 test utterances.

6.1.7. Evaluation platform

The experiments were conducted on a 2.4 GHz Intel Core2 based machine with an NVIDIA 8800GTX graphics processor. The operating system was Linux (Fedora Core 7 32 bit) and the decoder was compiled using the GCC compiler against CUDA toolkit version 1.1.

6.1.8. Baseline system

We compared the performance of the GPU accelerated decoder against two different CPU baseline implementations. The first baseline used a full logsum making use of the standard `log` and `exp` functions. SSE instructions were also used and at each frame the acoustic scores were evaluated on-demand and the score cached for the current frame. This baseline illustrates how the logsum can be performed rapidly on the GPU and to validate the accuracy of the GPU implementation.

The second baseline provided an alternative faster CPU comparison. We implemented several speed-up techniques and considered how they performed in combination. The final baseline substituted the logsum function for a logmax approximation in Eq. (11), this alone yielded significant speed-ups with a very small reduction in recognition performance.

$$\log p(\mathbf{x}) \approx \max_j \log \mathcal{N}(\mathbf{x}|\theta_j) \tag{11}$$

The faster baseline also used SSE instructions and an additional frame caching scheme or likelihood batch strategy as described in (Saraclar et al., 2002). When a state score is requested for the current frame the likelihood batching scheme will compute the scores for an additional block of K future frames and cache the results. The justification from (Saraclar et al., 2002) is that the memory bandwidth is the bottleneck not the actual computation. When a state is activated it is often requested again in the subsequent frames. Once the parameters have been moved from memory is it cheaper to perform future computation whilst the data is nearer the CPU and then cache the results.

Dimensional Gaussian pruning (Lee et al., 2001) did not bring any further speed-ups once the logmax and SSE heuristics were added. We also implemented a multi-core scheme which split the acoustic computations across two cores. Multi-core processors are often fed by the same memory bandwidth as their single-core counterparts and because the computation is bound by a memory bandwidth we were unable to obtain any further useful speed-ups.

6.2. Experimental results

6.2.1. Evaluation of window size on GPU performance

A major difference between the GPU and CPU scoring schemes is in the CPU implementation we only calculate and cache scores for states that are explicitly requested during search, whereas in the case of the GPU we always compute every state score for every frame regardless. Using the GPU in an on-demand approach will yield poor performance because the communication costs outweigh the computational power. The massive parallelism of the GPU means that not only calculating the score for every more efficient but calculating the score for every Gaussian in an entire block of samples yields even better performance. To illustrate this relationship we first evaluated how the speed characteristics of the decoder changed when adjusting the window size of samples sent to the GPU in each batch. For all decoding runs the search beams and bands were locked at 175 and 10,000, respectively and two sizes of acoustic models were considered with either 16 or 64 Gaussians per mixture model. For this experiment synchronous operation of the GPU was used. Fig. 1 illustrates the GPU decoder's RTF as the sample window size is increased. Moving from calculating a single frame to just a few frames per device transaction yields significant speed increases, as the window size is gradually increased the benefits from batching become smaller until convergence at around 30 samples per window. This window size is also convenient because it is close to the frequency of the garbage collector used in search and therefore the two could be synchronized when used in live decoding modes. This result clearly shows when using the GPU it is hugely beneficial to try and batch smaller transfers together as it will reduce the number of kernel launches on the device and the fewer synchronization points between the CPU and GPU aids the asynchronous operation.

6.2.2. Evaluation of likelihood batch size on CPU performance

In this section, we consider how the decoding speed changes as the CPU likelihood batch size is varied. We considered 16 and 64 components mixtures using a band of 1000 and a beam width 175 with logmax approximation and SSE enabled. Fig. 2 shows the RTF as the batch size is increased, based on these results we selected a batch size of 10 consecutive frames for the subsequent evaluations. The batch is slightly larger than the eight frames used in Saraclar et al. (2002) and Chen et al. (2006).

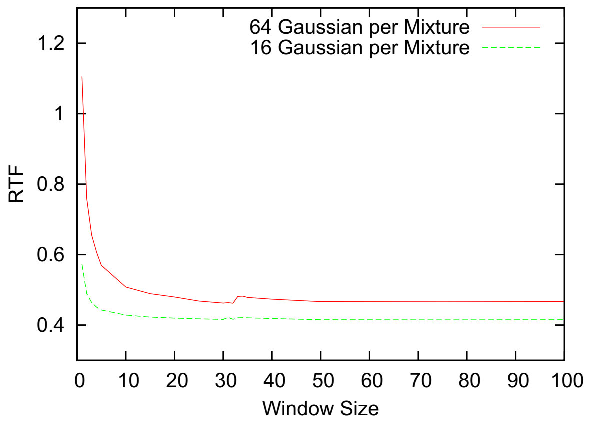


Fig. 1. Relation between the GPU sample window size and the speed of the GPU decoder for 16 and 64 component Gaussian mixture models.

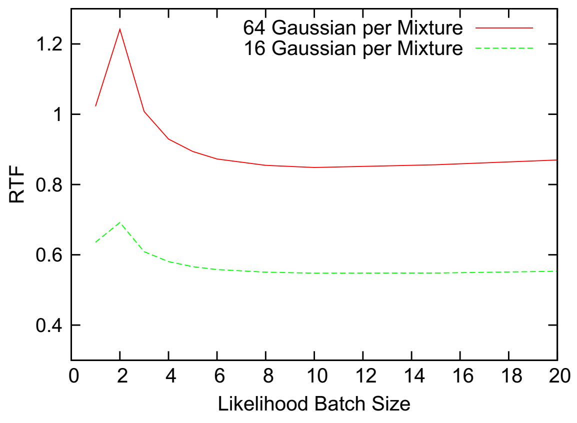


Fig. 2. Relation between the CPU likelihood batch size and the speed of the CPU decoder for 16 and 64 component Gaussian mixture models.

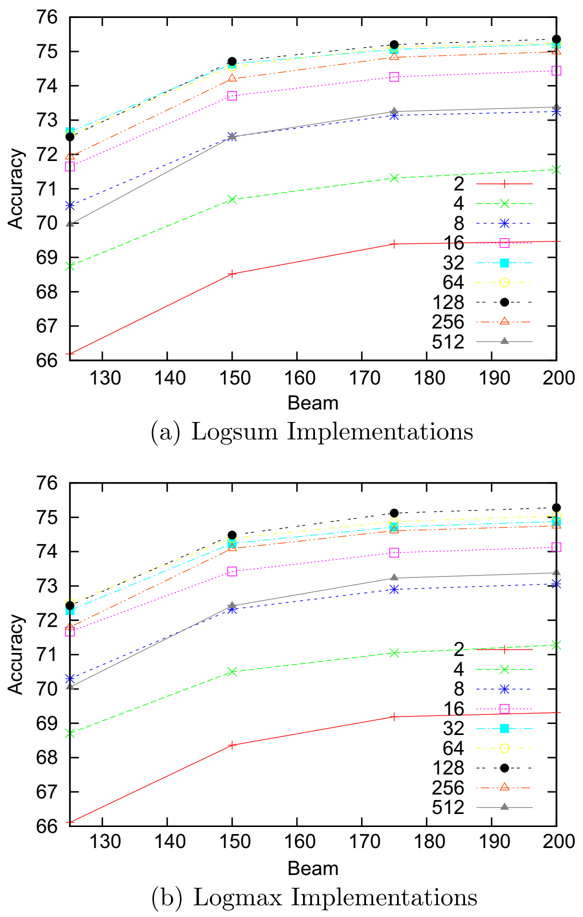


Fig. 3. Relation between beam width and accuracy for the logsum (a) and logmax (b) implementations on mixture sizes from 2 to 512. As expected the GPU and CPU techniques obtain the same accuracy for the same common parameters in both cases. This indicates both approaches are computing scores consistently.

6.2.3. Speed and accuracy

Next to illustrate the performance characteristics when using GPU acceleration, the GPU decoder was evaluated against the two baseline systems. The band was set to 10,000 for beam widths of 125, 150, 175 and 200 and the window size to 32 in all cases.

We first consider the accuracy of the various implementations when configured with the same models and search parameters. Fig. 3a shows the accuracy vs. beam width curves for either the GPU or CPU for logsum systems. We found that the GPU and CPU side equivalents consistently generated identical accuracy and utterance likelihood scores. As expected the use of the GPU does not degrade the accuracy and there do not appear to be any differences in accuracy caused by different host and device floating point implementations. Fig. 3b is the accuracy of the system using either the CPU or GPU logmax implementations, again both implementations achieve score identical accuracy. The logmax approximation leads to about 0.2% reduction in accuracy when compared to the full sum.

In Fig. 4 the accuracy vs RTF curves for the logsum implementations show the GPU accelerated decoder is substantially faster than the CPU decoder. The closer bunching of the GPU curves indicates increasing the mixture model is much less costly on the GPU than compared with the CPU counterpart. We achieved best accuracy for the 128 components mixtures, however the GPU could decode the larger component mixture models with practically no slow-down given the same search parameters

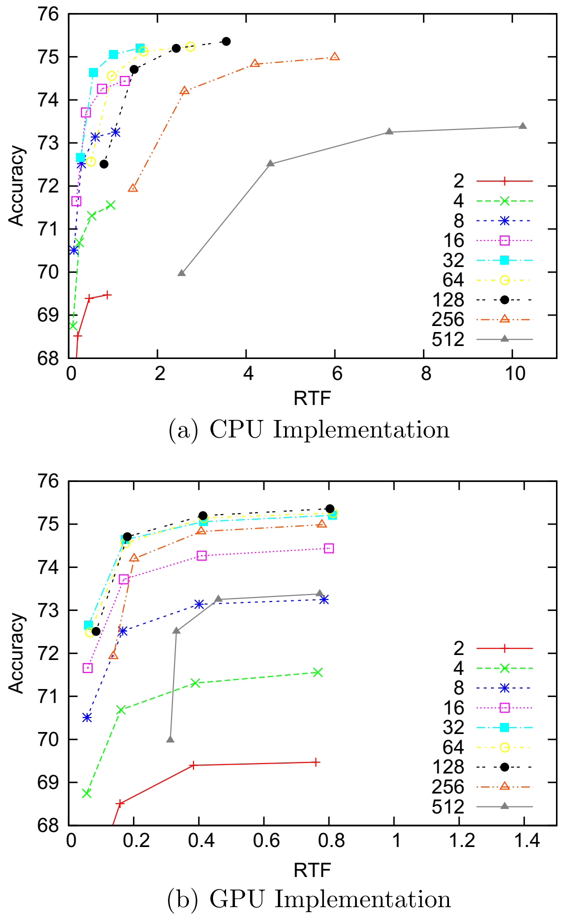
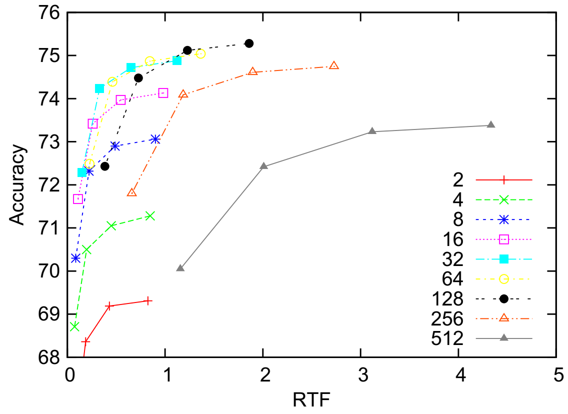


Fig. 4. Accuracy vs RTF for the CPU (a) and GPU (b) based acoustic model computations on mixture sizes from 2 to 512. When common search parameters are identical, both techniques achieve the same accuracy, however the curves in (b) clearly show the GPU is faster. The speed-up becomes more substantial for larger mixture models (note the different x-axis scales).

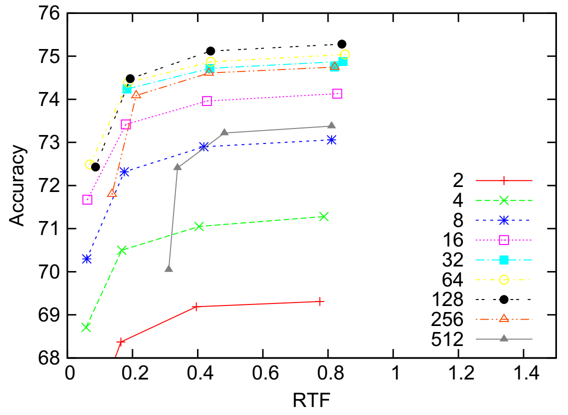
Fig. 5 shows the accuracy vs RTF curves when compared for the logmax optimized baseline. In this comparison, the GPU's logsum kernel was also substituted for a logmax kernel. The logmax CPU system with optimizations is substantially faster than the previous logsum CPU system. For the GPU system the logmax is slightly slower than the logsum, this is because the logsum does not present a computation bottleneck to the GPU and the better logsum scores may be aiding the pruning used in the main search procedure.

The Fig. 6 shows the speed-up factor and percent reduction in decoding time when enabling GPU acceleration compared to the optimized baseline when configured with same search parameters. The GPU decoder is faster than the CPU approach for all model sizes and beams widths that were evaluated, which is one of the major advances over the technique presented in Dixon et al. (2007). For the search settings which achieved best recognition accuracies the GPU enabled decoder was around 2.5–3 times faster. A larger reduction in decoding times is possible when considering the 512 component mixtures. However, on this task these larger models lead to a reduction in accuracy.

The next Fig. 7 further reinforces the benefits of using the GPU. The figure shows the RTF as the model complexity is increased for the different beam widths and the nearly flat GPU curves show there is very little reduction in speed from using large acoustic models, which is one of the main advantages of using the GPU for acoustic scoring.

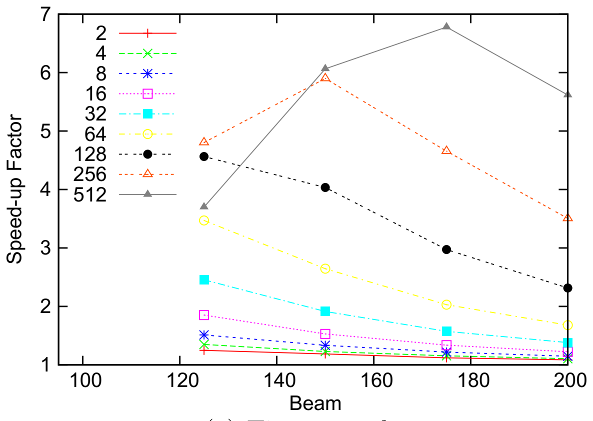


(a) CPU Implementation

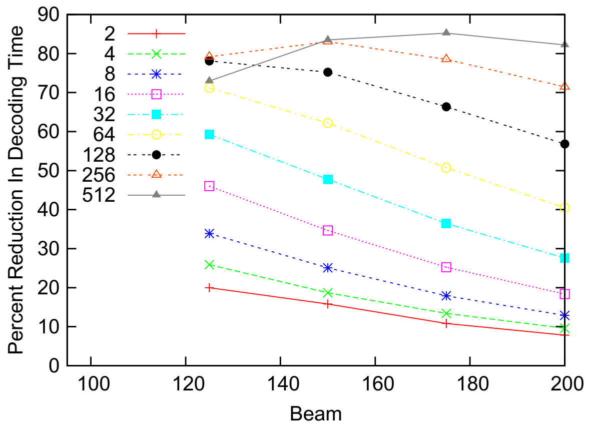


(b) GPU Implementation

Fig. 5. Accuracy vs RTF for the optimized logmax CPU (a) and logmax GPU (b) based acoustic model computations on mixture sizes from 2 to 512. Again the GPU scheme provides substantial speed-ups for large models (note the different x-axis scales).



(a) Times speed-up



(b) Percent reduction in decoding time

Fig. 6. When considering the relative speed-up of the GPU to the optimized logmax CPU implementation two metrics are compared. The times speed-up (Eq. (9)) shown in (a) and percent reduction in RTF (Eq. (10)) shown in (b).

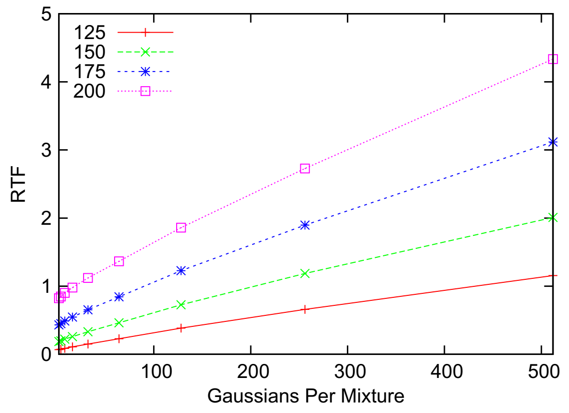
6.2.4. Percent of CPU time spent on acoustic scoring

The experiments were re-run with profiling enabled to measure the amount of CPU time spent executing the acoustic score computation and GPU co-ordination code. Fig. 8 shows that at beam widths which correspond to a convergence of accuracy only around 2–3% of the CPU time is used for the acoustic scoring related code. This is a very small amount of the overall CPU usage and in our decoder the computation which traditionally accounts of the largest amount of CPU time now requires a small fraction of the CPU resources. Fig. 9 shows the amount of time spent on acoustic model computation for the optimized logmax CPU baseline.

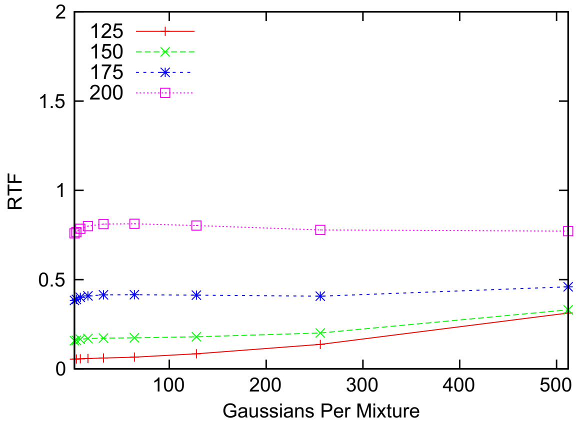
Moving the acoustic model computations to the GPU could also allow for super-linear like speed-ups to occur. In parallel computing a speed-up of greater than the number of processors can sometimes occur because of the changes in the caching and memory structure of the final system. By moving the acoustic computation from the CPU to GPU we have also removed more load off the main system memory buses and CPU cache. Previously memory bandwidth and CPU cache would have been used for moving and caching model parameters. Now these systems resources can be used for other parts of the decoder such as caching more of the compiled search network or active graph.

6.2.5. Power requirements

Because large speed-ups are possible by utilizing the GPU corresponding reductions in hardware size and costs may be possible. The final set of results shows the power consumption characteristics of the GPU and CPU decoders across the different sized models.



(a) CPU Implementation



(b) GPU Implementation

Fig. 7. RTF vs GMM size for the CPU (a) and GPU (b) each curve corresponds to a fixed set of search parameters with the beam width shown in the key. These results show for the GPU system there is very little overhead in increasing the mixture model complexity.

The test machine contained a dual core CPU but in these evaluations the single threaded version of our decoder was used, therefore, when the decoder is running it will not achieve 100% CPU usage on both cores and thus we would not expect it to reach the maximum power consumption of the CPU. Likewise with the

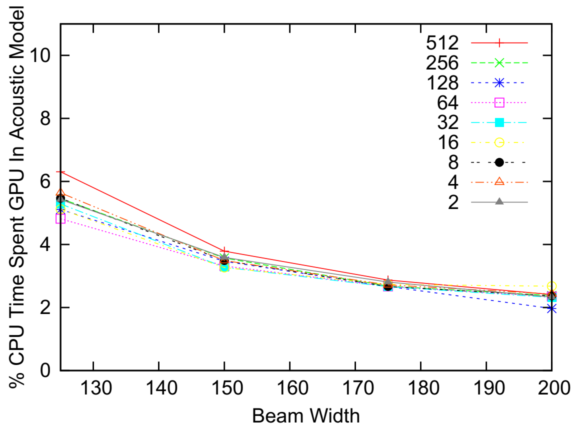


Fig. 8. Total percent of CPU time spent in acoustic model calculation code for the GPU accelerated decoder.

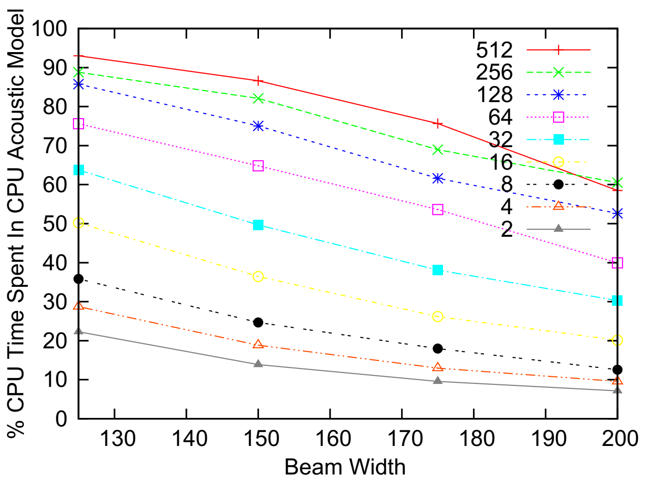


Fig. 9. Total percent of CPU time spent in acoustic model calculation code for the optimized CPU decoder.

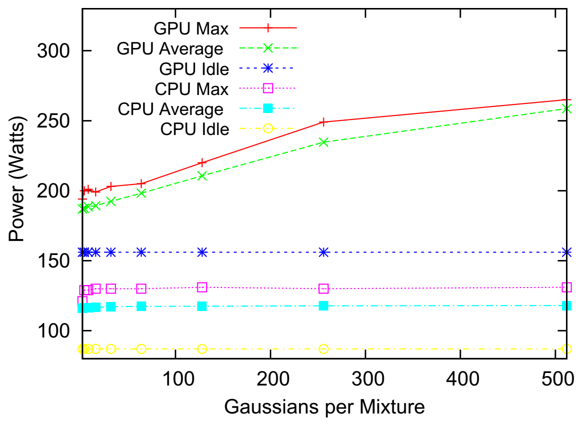


Fig. 10. Average and maximum power consumed when running the decoder with and without GPU acceleration.

GPU we would not expect to obtain maximum power consumption because even though all available cores will be utilized, CUDA is only using a subset of the GPU's hardware features as the graphics processing specific hardware is not used.

An inline power meter with logging was used to measure the power consumption of the entire machine. Power and clock frequency throttling were disabled and with the GPU installed we measured the machines idle consumption as 156 W. When measuring the power consumption of the pure CPU system the high-end GPU was substituted for an entry level video card to closer approximate the video hardware more likely to be found in a server machine. This lead to a substantial reduction in the idle power consumption to 86 W.

Fig. 10 shows the maximum and average power consumption for CPU and GPU decoders when using model sizes from 2 to 512. The beam and band widths were always set to 175 and 10,000, respectively. As expected the CPU power consumption is same regardless of model size, always averaging around the mid 170 W. When using the GPU decoder as the acoustic model size is increased we see the power consumption increasing, this is because as the model size gets larger more time is spent executing code on the GPU. The idle power consumption for the high-end GPU was higher than expected, however, more recent GPUs are reported to have a much lower idle power consumption.

7. Conclusions and future work

In this paper, we have described our GPU accelerated speech decoder. We have shown how to use a GPU co-processor in a way that leads to a very significant speed-up when compared to the CPU based techniques. The presented technique removes most of the compute intensive acoustic scoring off the CPU and allows for the use of very large acoustic models with only small slow-down in decoding. Furthermore the approach does not reduce the accuracy of the decoder.

We will continue improving our GPGPU and CPU scheme and to find the optimal techniques to obtain the best performance from the respective architectures. We would like evaluate the GPU technique on a task that achieves higher accuracy with 512 component and larger mixture models. Currently, the larger acoustic models are pushing the memory limits of the current generation GPUs to evaluate much larger models or use cards with less memory techniques to reduce memory computation would also be needed.

In the future we would like to investigate if the entire decoder could be ported to run on graphics hardware as well as investigating ways to make best of other asymmetric architecture such as the cell processor where there has been recent work by (Liu et al., 2007; Chong et al., 2008) demonstrating simpler speech decoders executing on these architectures. It is hoped the techniques presented in this paper can be adapted to other architectures or tasks. We would like to evaluate the suitability of GPUs for other speech processing tasks. For example the model training procedure where recent work by Shi et al. (2008) has shown GPUs achieving impressive speed-ups in clustering.

Acknowledgement

This work was supported by the Japanese government METI Project Development of Fundamental Speech Recognition Technology. The authors wish to thank Koji Iwano and the three anonymous reviewers for helpful comments and criticism.

References

Aubert, X., 1989. Fast look-ahead pruning strategies in continuous speech recognition. In: Proceedings of ICASSP. pp. 659–662.

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

Chan, A., Mosur, R., Rudnicky, A., Sherwani, J., 2004. Four-layer categorization scheme of fast GMM computation techniques in large vocabulary continuous speech recognition systems. In: Proceedings of INTERSPEECH. pp. 689–692.

Chen, S.F., Kingsbury, B., Mangu, L., Povey, D., Saon, G., Soltau, H., Zweig, G., 2006. Advances in speech transcription at IBM under the DARPA EARS program. IEEE Transactions on Audio, Speech, and Language Processing 14 (5), 1596–1608.

Chong, J., Yi, Y., Faria, A., Satish, N.R., Keutzer, K., 2008. Data-parallel Large Vocabulary Continuous Speech Recognition on Graphics Processors. Technical Report UCB/EECS-2008-69, EECS Department, University of California, Berkeley.

Dixon, P.R., Caseiro, D.A., Oonishi, T., Furui, S., 2007. The Titech large vocabulary WFST speech recognition system. In: Proceedings of ASRU. pp. 1301–1304.

Furui, S., 1986. Speaker-independent isolated word recognition using dynamic features of speech spectrum. IEEE Transactions on Acoustics, Speech, and Signal Processing 34 (1), 52–59.

Goffin, V., Allauzen, C., Bocchieri, E., Hakkani-Tur, D., Ljolje, A., Parthasarathy, S., Rahim, M., Riccardi, G., Saraclar, M., 2005. The AT&T WATSON speech recognizer. In: Proceedings of ICASSP. pp. 1033–1036.

Gschwind, M., Hofstee, H., Flachs, B., Hopkins, M., Watanabe, Y., Yamazaki, T., 2006. Synergistic processing in cell's multicore architecture. Micro, IEEE 26 (2), 10–24.

Hori, T., Nakamura, A., 2005. Generalized fast on-the-fly composition algorithm for WFST-based speech recognition. In: Proceedings of INTERSPEECH. pp. 847–850.

Kawahara, T., Nanjo, H., Shinozaki, T., Furui, S., 2003. Benchmark test for speech recognition using the corpus of spontaneous Japanese. In: Proceedings of ISCA and IEEE Workshop Spontaneous Speech Processing and Recognition. pp. 135–138.

Knill, K., Gales, M., Young, S.J., 1996. Use of gaussian selection in large vocabulary continuous speech recognition using HMMs. In: Proceedings of ICSLP. pp. 470–473.

Lee, A., Kawahara, T., Shikano, K., 2001. Julius – an open source real-time large vocabulary recognition engine. In: Proceedings of EUROSPEECH. pp. 1691–1694.

Liu, Y., Jones, H., Vaidya, S., Perrone, M., Tydlitát, B., Nanda, A.K., 2007. Speech recognition systems on the cell broadband engine processor. IBM Journal of Research and Development 51 (5), 583–591.

Maekawa, K., 2003. Corpus of spontaneous Japanese: its design and evaluation. In: Proceedings of ISCA and IEEE Workshop Spontaneous Speech Processing and Recognition. pp. 7–12.

Manocha, D., 2005. General-purpose computations using graphics processors. Computer 38, 85–88.

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

NVIDIA Corporation, 2007. NVIDIA CUDA Compute Unified Device Architecture Programming Guide.

Oonishi, T., Dixon, P.R., Iwano, K., Furui, S., 2008. Implementation and evaluation of fast on-the-fly WFST composition algorithms. In: Proceedings of INTERSPEECH. pp. 2110–2113.

Owen, G.S., Zhu, Y., Chastine, J., Payne, B.R., 2005. Teaching programmable shaders: lightweight versus heavyweight approach. In: SIGGRAPH'05: ACM SIGGRAPH 2005 Educators Program. p. 40.

Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Kruger, J., Lefohn, A.E., Purcell, T.J., 2005. A survey of general-purpose computation on graphics hardware. In: Eurographics 2005, State of the Art Reports. pp. 21–51.

Saon, G., Zweig, G., Kingsbury, B., Mangu, L., Chaudhar, U., 2003. An architecture for rapid decoding of large vocabulary conversational speech. In: Proceedings of Eurospeech. pp. 1977–1980.

Saon, G., Povey, D., Zweig, G., 2005. Anatomy of an extremely fast LVCSR decoder. In: Proceedings of Interspeech. pp. 549–552.

Saraclar, M., Riley, M., Bocchieri, E., Goffin, V., 2002. Towards automatic closed captioning: Low latency real time broadcast news transcription. In: Proceedings of ICSLP. pp. 1741–1744.

Shi, Y., Seide, F., Soong, F.K., 2008. GPU-accelerated gaussian clustering for fmpe discriminative training. In: Proceedings of Interspeech. pp. 689–692.

Volkov, V., Demmel, J.W., 2008. Benchmarking GPUs to tune dense linear algebra. In: SC'08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing. pp. 1–11.

Young, S., Russell, N., Thornton, J., 1989. Token Passing: A Simple Conceptual Model for Connected Speech Recognition Systems. Technical Report, Cambridge University Engineering Department.