FAST ACOUSTIC COMPUTATIONS USING GRAPHICS PROCESSORS

Paul R. Dixon　*Tasuku Oonishi*　*Sadaoki Furui*

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

ABSTRACT

In this paper we present a fast method for computing acoustic likelihoods that makes use of a Graphics Processing Unit (GPU). After enabling the GPU acceleration the main processor runtime dedicated to acoustic scoring tasks is reduced from the largest consumer to just a few percent even when using mixture models with a large number of Gaussian components. The results show a large reduction in decoding time with no change in accuracy and we also show by using a 16bit half precision floating point format for the acoustic model parameters, storage requirements can be halved with no reduction in accuracy.

***Index Terms*—** Speech recognition, LVCSR, GPGPU, WFST

1. INTRODUCTION

Acoustic scoring is the often most computational intense component of a large vocabulary speech decoder. Much previous research has focused on techniques to reduce or speed-up these calculations for example, using clustering, pruning, vectorization or look-aheads using less complicated models [1, 2, 3, 4, 5].

In this paper we describe our approach to fast acoustic scoring that moves the task completely off the CPU to execute on the Graphics Processor Unit (GPU). Modern graphics GPUs have evolved into a massively parallel processor with phenomenal floating point throughput and memory bandwidth. The field of *General Purpose computations on GPUs* (GPGPU) emerged to allow non-graphics applications to harness this massive computational resource. GPGPU was initially a very complicated task because of the restrictive GPU programming model, however, the introduction of the Compute Unified Device Architectures (CUDA) framework by NVIDIA has simplified the development allowing GPGPU to become a more mainstream tool. GPGPU has been successful in many other scientific and engineering applications. In [6] we first proposed a GPGPU proof of concept technique for acoustic scoring in speech recognition.

The technique we introduce in this paper represents a significant advance in the performance of our decoder. We first describe our method that allows for the acoustic computation to be moved to the GPU. The scheme allows for concurrent utilization of both the GPU and CPU during the recognition process. Thorough evaluations are presented on a large vocabulary spontaneous task using a set of models of varying complexity. The results demonstrate a speed-up in all cases with the most significant speed-ups occurring for mixture models with large numbers of Gaussians. We show by using the GPU it is possible to increase the number of Gaussians in the system by two orders of magnitude whilst incurring practically no slow down. Finally, we demonstrate the soon to be standard half floating point format allows more efficient memory and bandwidth usage with no loss in recognition accuracy.

2. GPU HARDWARE AND CUDA

In these evaluations 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 multiprocessors cores executes the same instruction stream in parallel operating on different portions of data. CUDA provides a `C` like language for writing *kernels*, these are the functions that execute on the GPU. Functions are also provided for allocating buffers on the graphics cards and transferring data between the graphics and main memory. The PCI-Express x16 bus to the GPU provides a theoretical peak transfer rate of 4GB/s in each direction. Under the CUDA framework the GPU becomes a massively threaded co-processor. 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). 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 hardware such as branch prediction obtaining performance gains on code containing a large amount of conditional logic is more difficult.

3. ACOUSTIC SCORING USING GPUS

Our GPGPU approach is a hybrid scheme that uses the CPU and GPU to perform different computations during the speech decoding process. The rationale behind the design is to target the GPU for floating point and memory bandwidth intense acoustic computations and the CPU to perform the part of the search algorithm that involves dynamic data structures and frequent control flow operations. Through concurrent operation we also ensure both processors are kept working as much as possible.

3.1. Approach to Likelihood Computations

In the GPU approach the likelihood computation is performed as a matrix multiplication. The acoustic parameter matrix $\mathbf{A}$ will have one row for every Gaussian in every state, leading to a total row count of states $\times$ Gaussians' per state. Given a state the i^{th} the J dimensional weighted Gaussian with diagonal covariance is given by:

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

This can be expressed as a row 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 w_i is the weight, μ_{ij} and σ_{ij}^2 are the mean and variance in the j^{th} dimension and K_i is:

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

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:

$$\mathbf{z}^T = \left\{1, \mathbf{x}_1, \ldots, \mathbf{x}_J, \mathbf{x}_1^2, \ldots, \mathbf{x}_J^2\right\} \tag{4}$$

and then performing the matrix vector multiplication $\mathbf{y} = \mathbf{Az}$ [7]. The result is a vector $\mathbf{y}$ containing log weighted scores of the feature vector for every Gaussian component for every state mixture model.

3.2. Performance Enhancements

After moving the Gaussian computations to the GPU the following enhancements are also used to maximize the GPU usage and address the communications bottleneck between the CPU and GPU.

Batching smaller transfers into larger transfers reduces the frequency of calls to the GPU. This involves sending windows of acoustic frames to the GPU for calculation. On the device we now receive a matrix W that contains a windows of n feature vectors:

$$\mathbf{W} = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_n\} \tag{5}$$

The scores for every Gaussian in every mixture for the entire window are computed using the matrix multiplication $\mathbf{F} = \mathbf{AW}$, each column in $\mathbf{F}$ contains the log weighted scores for every Gaussian component of every mixture model for the corresponding feature vector in $\mathbf{W}$. Another advantage of the batching approach is the use of the faster matrix-matrix multiplication `sgemm` that performs more floating point operations per memory move. For this a highly tuned matrix multiply implementation [8] was used.

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

$$\log p(\mathbf{x}) = \log\left(\sum_i w_i p(\mathbf{x} \mid \theta_i)\right) \tag{6}$$

Because we are dealing with the log weighted Gaussian scores $\log w_j p_j(\mathbf{x} \mid \theta_j)$ 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 in $\mathbf{F}$ and a reduction pattern is used to maximize performance. 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 because we have all 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[7, 3] as the acoustic scores are accessed during the search.

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. By always using page-locked memory for the host transfer buffers it should be possible to obtain further speed-ups.

When the GPU is computing acoustic scores the CPU is idle waiting for the kernels to complete. If search and acoustic scoring could occur in parallel further performance benefits could be achieved by hiding the GPU compute latencies. To remove the need for a multi-threaded solution we make use of the *streams* and *asynchronous* launches available in CUDA 1.1[9]. The CUDA stream objects allow multiple kernels or CUDA functions to be grouped together into a single unit which can be launched from a single invocation. When a function is launched asynchronously control will return to the CPU immediately, the CPU can poll the GPU to find out whether the function has completed.

3.3. Decoder Integration

Two methods are exposed to the decoder to allow for asynchronous operation. A *compute* function that will launch a stream to perform the following: Transfer sample window to GPU; Compute Gaussian scores as matrix multiplication; Compute mixture scores using logsum; Transfer scores back to host.

A *wait* method is used to query the GPU for computation condition and act as a barrier. The GPGPU implementation was integrated into the large vocabulary decoder currently under development at Tokyo Institute of Technology[6]. The decoding engine is a time-synchronous Viterbi beam system that operates on Weighted Finite State Transducer (WFST)[10] search spaces. The decoder can use the *compute* method to calculate state scores for windows of feature vectors. After the first window has been computed, the decoder can launch the next window compute asynchronously, allowing the CPU to decode the current window scores simultaneously. This asynchronous pattern is repeated until all features have been computed and decoded. The wait method is used to enforce correct synchronization between the CPU and GPU.

3.4. Half Precision Floating Point

The decoder uses single precision floating point representation and we have found speech decoding does not benefit from using the 64 bit double precision format. The revised IEEE floating point specification also describes a 16bit half precision format which has a sign bit, five bit exponent and a 10 bit mantissa. Currently CUDA has limited hardware 16bit float support for texture stored via the driver level API [9]. Future versions of CUDA may provide more flexible usage of the 16bit floats, also the upcoming *Larrabee* processor [11] will also support storage and hardware conversion of 16bit floating point. Fixed point representations have already been investigated to reduce memory or computation for example in [12], however, half floats are interesting because of the potential for standard hardware support. To evaluate the suitability of this new format a software implementation is used to investigate the accuracy of the GPU accelerated decoder when storing acoustic model parameters as 16bit floats.

4. EVALUATIONS

4.1. Experimental Setup

4.1.1. Corpus and Test Set

Our evaluations were done on the Corpus of Spontaneous Japanese (CSJ). The 10 lecture test-set was segmented into 2328 utterances.

4.1.2. Acoustic Features

The original 16kHz raw speech was first segmented into utterances, then converted to a 39 dimensional Mel-frequency cepstral coefficients (MFCC) based parametrizations with a 10ms frame rate and 25ms window size. Each feature vector was composed of 12 MFCC values and an energy term with delta and delta–delta values. The energy term was then discarded to give the final 38 dimensional feature representations.

4.1.3. Models

A set of acoustic models of various complexities were built to allow us to investigate the speed characteristics of the GPU accelerated decoder. The topology of all the HMM acoustic models was a three state left-to-right tri-phone model. The state output mixture models were built using an Expectation-Maximization (EM) algorithm with mixture model splitting. There were eight sets of acoustic models each with 3000 states, the mixture models contained power-of-two sizes from two to 512 Gaussian components per mixture each with diagonal covariance.

The language model was a back-off trigram using Katz discounting with a vocabulary of 65k words trained using the CSJ training data. The knowledge sources were used to create a $H \circ C \circ L \circ G$ recognition cascade, where H represents the acoustic models, C the context-dependency, L is the lexicon and G is the language model. The factored arcs were dynamically expanded in the decoder.

4.1.4. Evaluation Platform

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

4.2. Results

We compared the GPU accelerated system with a window size of 32 to a baseline CPU implementation that performed on demand acoustic computations, made use of the CPU vector instructions, and substituted the logsum of equation 6 with a *logmax* approximation that selects only the top scoring Gaussian.

Figures 1 and 2 shows the Real Time Factor (RTF) vs accuracy curves for both the CPU and GPU accelerated decoders for beam widths of 100, 125, 150, 175 and 200. When using identical search parameters a very slight increase in accuracy is observed in the GPU system owing to the use of the full logsum. We also implemented a GPU logmax function and observed identical recognition accuracy as the CPU implementation. The speed of the GPU logmax and logsum implementations was identical, this is party because the GPU has fast logarithm and exponential implementations. Therefore, unlike the CPU approach another advantage of technique is the expensive logsum can be used on the GPU with no sacrifice in speed.

The GPU accelerated decoder is faster than the CPU approach for all model sizes and beams widths that were evaluated. To further illustrate this point Figure 3 shows the speed-up factor when enabling the GPU for the various parameter settings.

The closer bunching of the GPU curves indicates that increasing the model order is much less costly on the GPU system. For a wider beam increasing the model complexity has only a negligible reduction in decoding speed. This behaviour can be explained by the profiler output for the GPU evaluations in Figure 4 which shows the percent of CPU time spent in acoustic scoring related code. We see regardless of model size, typically only a very small percent of the CPU runtime is dedicated to acoustic scoring related tasks. Because of insufficient training material the accuracy for the 256 and 512 component mixtures reduced, the results were included to show that the GPU acceleration could use these large models with little speed penalty. In the GPU accelerated decoder what is traditionally the most computational demanding task has been reduced to just a few percent even for a significant amount of Gaussians.

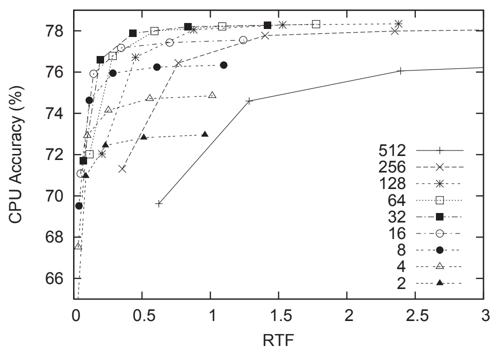

Fig. 1. RTF vs accuracy of the decoder when using CPU based acoustic scoring.

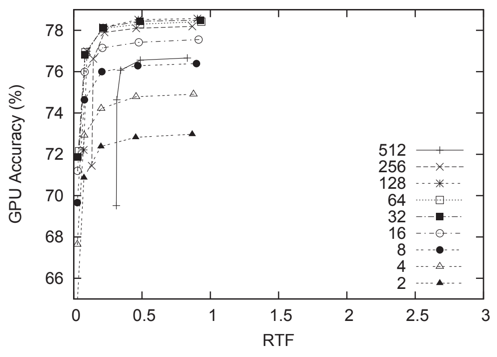

Fig. 2. RTF vs accuracy of the decoder when using GPU based acoustic scoring.

4.3. Discussion of Technique

After moving the acoustic scoring to the GPU, computing scores on a window of frames was most effective in obtaining the large speedups. When switching from a single frame to just a small window we saw the largest improvements, after a window size of approximately 30 frames the speed converged to constant value.

Using the asynchronous operation and allowing the decoder to simultaneously use the GPU and CPU gave the biggest speed-ups for the larger models of 128 Gaussians and greater. This indicates for the smaller models even on a window of frames the whole calculation is very fast and the CPU isn't stalled much waiting for GPU computations to complete.

Page locked buffers lead to a several percent reduction in RTF and was a simple improvement requiring only small code changes.

Running the logsum on the card gave a large speedup when compared to a CPU based logsum and using a card based logsum is no

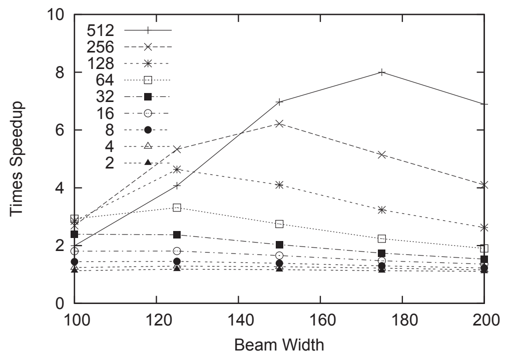

Fig. 3. Speed-up factor of the decoder after enabling GPU based acoustic scoring.

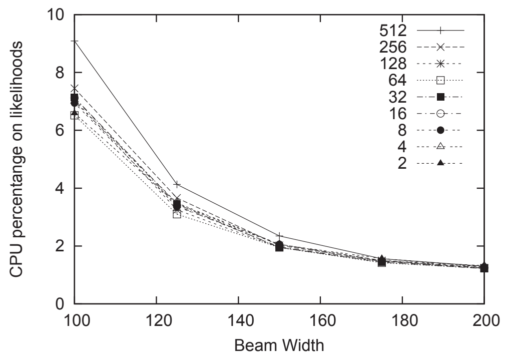

Fig. 4. Percent of CPU runtime dedicated to acoustic scoring after enabling GPU based acoustic scoring.

more expensive than a logmax. Either approach allows for the acoustic score cache to be replaced with a simple buffer.

4.4. Half Precision Floating Point Evaluations

The final evaluations illustrate the decoder's accuracy when storing parameters in half float format and using GPU acceleration. Table 4.4 shows the accuracy when using 16, 32 and 64 component mixture models with the parameters stored in either single or half floating point. In all cases the half float representation did not cause any major reduction in accuracy.

		Beam				
Precision	Gaussians	100	125	150	175	200
Single	16	71.2	75.99	77.15	77.42	77.55
	32	71.86	76.81	78.13	78.44	78.49
	64	72.15	76.94	78.09	78.3	78.42
Half	16	71.24	75.97	77.1	77.4	77.54
	32	71.9	76.81	78.12	78.44	78.49
	64	72.17	76.94	78.08	78.31	78.43

Table 1. Accuracy of the decoder when using half and single precision floating point storage for acoustic model parameters.

5. CONCLUSIONS

In this paper we have introduced our GPU acoustic computation scheme that gives a substantial speed-up by moving all the acoustic likelihood calculations off the CPU and does not cause any accuracy trade-offs. The approach can rapidly compute large mixture models and we show the GPU technique is especially well suited to tasks that demand large numbers of Gaussians. The implementation is highly parallel and will scale to future GPUs that have a larger number of cores. For future work we will attempt to move more of the decoder to execute on the graphics card. We have additionally verified the 16bit floating format point is sufficient for storing acoustic model parameters, halves memory requirements and has the potential for standardized hardware support.

6. REFERENCES

[1] X.L. Aubert, "Fast look-ahead pruning strategies in continuous speech recognition," in *Proc. ICASSP*, 1989, pp. 659–662.

[2] K. Knill, M. Gales, and S. J. Young, "Use of Gaussian selection in large vocabulary continuous speech recognition using HMMs," in *Proc. ICSLP*, 1996, pp. 470–473.

[3] G. Saon, G. Zweig, B. Kingsbury, L. Mangu, and U. Chaudhar, "An architecture for rapid decoding of large vocabulary conversational speech," in *Proc. EUROSPEECH*, 2003, pp. 1977–1980.

[4] A. Chan, R. Mosur, A. Rudnicky, and J. Sherwani, "Four-layer categorization scheme of fast GMM computation techniques in large vocabulary continuous speech recognition systems," in *Proc. INTERSPEECH*, 2004, pp. 689–692.

[5] V. Goffin, C. Allauzen, E. Bocchieri, D. Hakkani-Tur, A. Ljolje, S. Parthasarathy, M. Rahim, G. Riccardi, and M. Saraclar, "The AT&T WATSON speech recognizer," in *Proc. ICASSP*, 2005, pp. 1033–1036.

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

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

[8] V Volkov and J W. Demmel, "Benchmarking GPUs to tune dense linear algebra," in *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, 2008, pp. 1–11.

[9] NVIDIA Corporation, "NVIDIA CUDA compute unified device architecture programming guide," 2007.

[10] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech and Language*, vol. 16, no. 1, pp. 69–88, 2002.

[11] L Seiler, D Carmean, E Sprangle, T Forsyth, M Abrash, P Dubey, S Junkins, A Lake, J Sugerman, R Cavin, R Espasa, E Grochowski, T Juan, and P Hanrahan, "Larrabee: a many-core x86 architecture for visual computing," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 1–15, 2008.

[12] E. Bocchieri and D. Blewett, "A decoder for LVCSR based on fixed-point arithmetic," in *Proc. ICASSP*, 2006, pp. 1113–1116.