An Implementation of Decision Tree-Based Context Clustering on Graphics Processing Units

Nicholas Pilkington[1 2], *Heiga Zen*[1]

[1] Toshiba Research Europe Ltd., Cambridge Research Laboratory, Cambridge, UK
[2] Cambridge University Computer Laboratory, Cambridge, UK

Abstract

Decision tree-based context clustering is essential but time-consuming while building HMM-based speech synthesis systems. Its widely used implementation is not designed to take advantage of highly parallel architectures, such as GPUs. This paper shows an implementation of tree-based clustering for these highly parallel architectures. Experimental results showed that the new implementation running on GPUs was significantly faster than the conventional one running on CPUs.

Index Terms: HMM-based speech synthesis, decision tree-based context clustering, GPU

1. Introduction

Hidden Markov model (HMM)-based speech synthesis [1] has grown in popularity in recent years. In this framework, the spectrum, excitation, and durations of speech are modelled simultaneously in a unified framework of HMMs. For a given text to be synthesized, speech parameter trajectories that maximise their output probabilities are generated from estimated HMMs under constraints between static and dynamic features.

In both speech recognition and synthesis systems, context-dependent HMMs have widely been used. Triphone HMMs are typically used in speech recognition. In speech synthesis, segmental, prosodic and linguistic contexts in addition to phonetic contexts (sometimes called "fullcontexts") are often used. There are a huge number of possible combinations of contexts and it is almost impossible to cover all possible combinations of contexts with a finite set of training data. To address this problem, the decision tree-based context clustering technique was proposed [2]. This technique clusters similar HMM states (or streams) into the same class based on their contexts using pre-defined questions about the contexts. Then, it ties model parameters among HMM states (or streams) associated with the same class. This technique allows for robust parameter estimation as well as decreasing the number of unique models that have to be dealt with subsequently. By introducing the prior knowledge about context into the pre-defined questions, it also offers generation of unseen contexts by assigning them to a certain class by traversing decision trees. For building HMM-based speech synthesis systems, a software toolkit called HTS is often used [3], which is released as patch code for the hidden Markov model toolkit (HTK) [4]. An implementation of the decision tree-based context clustering technique has been included in HTK (and thus HTS) for many years and is in common use.

The graphics processing units (GPUs) are specialized for compute-intensive, highly parallel computation because of its birth in graphics rendering. This contrasts with central processing units (CPUs) which are concerned with high speed sequential processing and data locality. In the last few years the computational power of GPUs has overtaken that of CPUs and continues to increase. In addition to the main CPU, almost every computer is equipped with a GPU which is in essence a specialized parallel processor. A GPU is mainly a single instruction, multiple data (SIMD) parallel processor that is computationally powerful, while still being quite affordable. The advent of many-core GPUs means that mainstream processor chips are now parallel systems. Furthermore, their parallelism continues to scale with Moore's law. A noteworthy technological advance was achieved in 2007, when NVIDIA introduced the compute uniform device architecture (CUDA). This greatly enhanced the flexibility and usability of the GPU, to the extent that it is becoming a mainstream alternative for general purpose calculations [5]. It has been utilized in speech recognition [6, 7].

Table 1: *Training time for building a speaker-dependent HMM-based speech synthesis system.*

Training part	**Training time (sec.)**
Initialization & reestimation	1,223
Embedded reestimation	9,117
Tree-based context clustering	40,170

Training of HMM-based speech synthesis systems consists of three parts; initialization, embedded reestimation and decision tree-based context clustering. Table 1 shows the training time for building a speaker-dependent HMM-based speech synthesis system with hours of speech data. It can be seen from the table that decision tree-based context clustering takes up a large portion of the training time (about 80%) and is indeed the most time consuming operation.[1] The implementation of the decision tree-based context clustering that exists in HTK/HTS is not designed to take advantage of highly parallel architectures. Decreasing the time required would speed up the entire process and decrease the turnaround time of making experimental changes and realising the results during development. This paper presents an implementation of the decision tree-based context clustering technique on GPUs based on CUDA.

The rest of this paper is organized as follows. Section 2 gives an overview of decision tree-based context clustering. Section 3 describes the GPU architecture. Section 4 presents the implementation of decision tree-based context clustering using GPUs. Section 5 shows performance comparison between the conventional and proposed implementations. Concluding remarks and future plans are presented in the final section.

[1] Embedded reestimation was distributed to 10 CPUs and processed in parallel. Decision tree-based context clustering was also distributed to 16 CPUs and processed in parallel. Based on the current HTK implementation, further parallelization of the decision tree-based clustering process is not available.

2. Decision Tree-Based Context Clustering

In the decision tree-based context clustering technique [2], a top-down clustering is performed so as to locally maximize the log likelihood of models to the training data using pre-defined questions about contexts. Then, mean vectors and covariance matrices of HMM states (or streams) clustered to the same leaf (terminal) node are tied.[2] As a result, the HMM state-level (or stream-level) tying structure can be constructed. The mean vector and the covariance matrix associated with the leaf node S, $\boldsymbol{\mu}_S$ and $\boldsymbol{\Sigma}_S$, can be estimated based on the ML criterion as

$$\hat{\boldsymbol{\mu}}_S = \frac{\sum_{t=1}^{T} \sum_{m \in \boldsymbol{M}_S} \gamma_m(t) \boldsymbol{o}_t}{\sum_{t=1}^{T} \sum_{m \in \boldsymbol{M}_S} \gamma_m(t)}, \quad (1)$$

$$\hat{\boldsymbol{\Sigma}}_S = \frac{\sum_{t=1}^{T} \sum_{m \in \boldsymbol{M}_S} \gamma_m(t) (\boldsymbol{o}_t - \hat{\boldsymbol{\mu}}_S)(\boldsymbol{o}_t - \hat{\boldsymbol{\mu}}_S)^\top}{\sum_{t=1}^{T} \sum_{m \in \boldsymbol{M}_S} \gamma_m(t)}, \quad (2)$$

where $\top$ denotes the matrix transpose, T is the total number of frames in the training data, $\boldsymbol{M}_S$ is a set of HMM states (or streams) clustered to the node S, and $\gamma_m(t)$ is the posterior probability of an HMM state (or stream) m for an observation vector at frame t, $\boldsymbol{o}_t$. The HTS implementation of the decision tree-based context clustering imposes a "variance floor" – a minimum value for estimating variance – in the clustering process. If values in $\hat{\boldsymbol{\Sigma}}_S$ are smaller than the variance floors, they are floored. Typically, variance floors are set as a fraction of the total empirical variance across all the training data.

The total log likelihood of the HMM state (or stream) of node S to the associated training data is calculated as

$$\mathcal{L}(S) = -\frac{1}{2} \sum_{t=1}^{T} \sum_{m \in \boldsymbol{M}_S} \gamma_m(t) \left\{ n + \log \left(2\pi \left| \hat{\boldsymbol{\Sigma}}_S \right| \right) \right\}, \quad (3)$$

where n is the dimensionality of $\boldsymbol{o}_t$. Extending the decision tree-based context clustering to the multi-space probability distribution HMM (MSD-HMM), which has been used to model F_0 observations consisting of voiced and unvoiced frames, has also been derived and implemented in HTS.

The minimum description length (MDL) criterion [8] has been used in the HMM-based speech synthesis system to automatically control the size of decision trees. When cluster S is divided to S_{q+} and S_{q-} by a question q, the change of total description length by this split, $\Delta_q^{(\mathrm{DL})}(S)$, is calculated as

$$\Delta_q^{(\mathrm{DL})}(S) = \mathcal{L}(S) - \{\mathcal{L}(S_{q-}) + \mathcal{L}(S_{q+})\} + \frac{N}{2} \log \left\{ \sum_{t=1}^{T} \sum_{m \in \boldsymbol{M}_{S_0}} \gamma_m(t) \right\}, \quad (4)$$

where S_0 denotes a root node of this decision tree and N is the number of parameters increased by this split. If covariance matrices are assumed to be diagonal, $N = 2n$.

3. GPU Architecture

Programming for the GPU provides different mechanisms to the CPU – most notably there are different execution and memory spaces. The inherent concurrency is also worth considering as it is easy to introduce race conditions on memory locations that are being accessed by more than one thread at a time.

[2] Usually, state-output (or stream-output) distributions of HMMs are assumed to be single Gaussian distribution with diagonal covariance matrices during the clustering process.

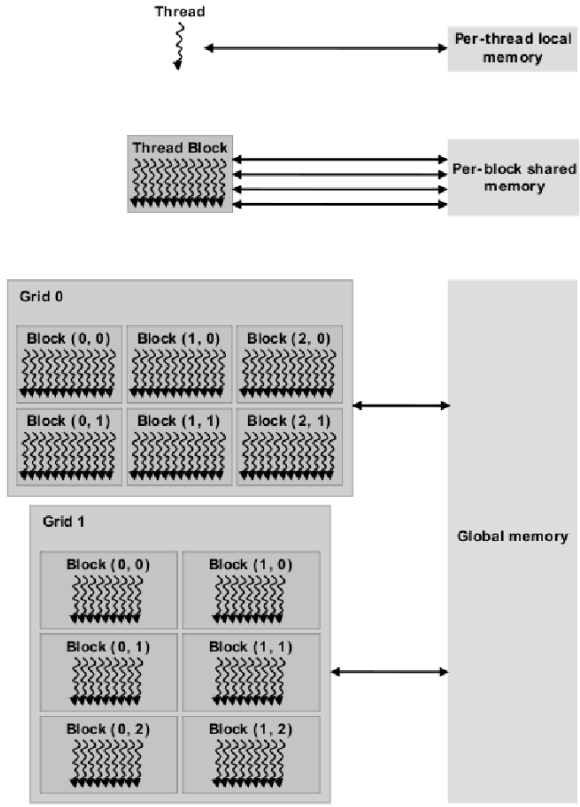

Figure 1: CUDA memory space arrangement.

3.1. Thread arrangement

The CUDA framework allows for the development of kernels which are pieces of code that are compiled and executed on the GPU in parallel by a number of CUDA threads. It organises CUDA threads into blocks of varying dimensions. Each thread is given an index within its block. Blocks are in turn organized into a multidimensional grid. Thread blocks are required to execute independently. It must be possible to execute them in any order, in parallel or in series. This independence requirement allows thread blocks to be scheduled in any order across any number of cores, enabling programmers to write code that scales with the number of cores. The number of thread blocks in a grid is typically dictated by the size of the data being processed rather than by the number of processors in the system, which it can greatly exceed.

3.2. Memory arrangement

There are three different memory spaces available to a CUDA thread. The first is private memory which is accessible solely by the thread in question. Then there is shared memory which is accessible to all threads that are in the same blocks but beyond the scope of other blocks. Finally there is global memory that can be accessed by all the threads at any time. These memory spaces are shown in Fig. 1. The CUDA framework also provides two read-only memory spaces which can be accessed by all threads. These are the constant memory and the texture memory spaces. Each of the global, constant and texture memory spaces is optimized for different access patterns [5].

4. Implementation

In order to implement decision tree-based context clustering on GPUs the problem must be reformulated in terms of data independent sections that can be executed in parallel. The information that needs to be accessible to the kernels and which parts of that information need to persist across kernels and be accessible by different threads concurrently must also be identified.

The implementation took the following form. Initially all the models are placed in a single cluster. The single question that gives the greatest increase in log likelihood in the sense of Eq. (3) to split this cluster then has to be found. Since the results of computing the increase in log likelihood of a cluster for a single specific question is independent of all the other questions it can be computed concurrently with all the other questions. This should give a larger performance gain as now the running time is reduced from all the possible question-cluster pairs being computed sequentially to all being done in parallel. In order to compute the increase in log likelihood of splitting a given cluster with a given question all the statistics both for the models that are congruent with the questions and those that are not have to be accumulated. This in turn requires that each executing thread has access to all the statistics and occupancy counts of the models in its memory space. This means that they must all be copied into the global GPU memory. A single thread can then be launched for each question and cluster pair to compute the increase in log likelihood. The basic flow of the algorithm was constructed as follows - each part will be discussed separately.

```
1   Preprocess()
2
3   while(NOT Threshold)
4   {
5      GPUPush();
6      SplitAllClusters();
7      GPUPull();
8   }
9
10  PostProcess();
```

The implementation comprises three stages. Initially the preprocessing stages copy all information that does not change during the execution of the program to the global GPU memory space. Then for a number of iterations until a threshold is met, all level dependent information is copied to the GPU overwriting the existing values. The best question to split each cluster is computed and these values are copied back to CPU memory space. Figure 2 illustrates the overview of the implementation. The following sections deal with each part of the algorithm in more technical detail.

4.1. `Preprocess()`

The decision tree-based context clustering technique requires the model statistics as well as a list of questions. In order to decrease the computational cost, the questions and model names were used to create a bitfield to efficiently look up whether a question and model were congruent. This bitfield was stored and used during a clustering process.

The preprocess was also concerned with once off operations that were required for the clusters but whose values persisted throughout the execution of the clustering. There are three datasets that the kernel needs to access throughout the entire clustering algorithm. The first is the model statistics. These

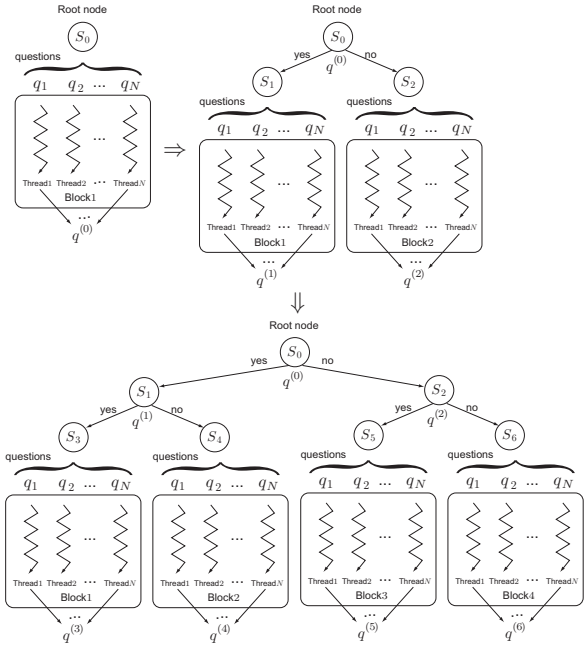

Figure 2: Overview of the proposed implementation of decision tree-based context clustering based on GPUs.

are the first order, second order and occupancy counts for each model in the stream and mixture being clustered. The second dataset is a structure to identify when a model is congruent with a question, as mentioned earlier this was precomputed for each distinct model and question set and stored in a two dimensional bitfield where the bit is set if the question corresponding to the first index is congruent with the model corresponding with the second, and cleared otherwise. These two datasets are allocated in global GPU memory. The third one is the variance flooring values. Like the HTS implementation, the implementation performs variance flooring in the clustering process. These values are accessed by all threads but constant within the entire clustering process and small. Therefore, this dataset is allocated in constant memory rather than global GPU memory. These three datasets are copied to their relevant memory spaces at the start of the execution.

4.2. `GPUPush()`

This function is concerned with moving the data to the GPU that is changed with each execution of the kernel and thus is not part of the preprocess but is updated on each iteration of the clustering. This information includes the cluster to which each model is currently assigned. The assignment information is not changed until after the kernel is finished executing, but it is required during each execution to determine which models are in which clusters. This information is allocated in global GPU memory as well.

4.3. `SplitClusters()`

This involves the actual execution of the kernel. As mentioned earlier the kernel is launched and executed once by each thread in each block. The kernel is launched on a one dimensional block whose dimension is the number of un-split clusters at the current depth in the decision tree. These are increasing powers

of two as initially there is one cluster, then after the split there are two, then four and so on. The total number of threads is always the same and is equal to the number of questions. Thus there is a kernel executing for each cluster and question.

The purpose of the kernel is to solely determine what the increase in log likelihood would be if that question was used to split that cluster. This is computed in a similar way to the HTS implementation. Firstly the thread accumulates the statistics for models that are congruent with the question (answer "yes") and for models which are not (answer "no"). It then computes the increase in log likelihood as Eq. (3) for the "yes" cluster and "no" cluster. The sum of these values minus the parent log likelihood gives the increase in log likelihood of the split. If this is better than the other increases seen so far the increase and the index of the question are stored.

4.4. `GPUPull()`

This function corresponds to the GPUPush and pulls the information from the GPU so it can be accessed on the host machine. This is the array of best questions for each cluster, the values of increases in log likelihood of each split, and finally the assignment of each model to a cluster. The actual increases in log likelihood are used to check whether they are beyond the threshold or not. Once all this information has been copied off the GPU, each model in each un-split cluster is assigned to a new cluster based on its best question. This is simply a check of congruency between the best question and the model as the best question for each cluster has already been determined in the kernel.

4.5. `Postprocess()`

Here the reconstruction of the actual decision tree from the cluster markers on each model and the list of best questions for each cluster at each level is performed. It should be noted that the node numberings produced by the GPU clusters are not necessarily the same as the node numbers of the HTK/HTS.

5. Experiment

Performance analysis was carried out with the dataset used in Table 1 on a machine which had two Intel Core i7 920 CPUs (8 cores in total) and one NVIDIA Tesla C1060 GPU. The speech analysis conditions and model topologies of Nitech-HTS 2005 [9] were used. All results reported here were measured with building 10 decision trees in serial[3] averaged over 10 trials. The number of distributions to be clustered was 195,273 and the number of questions about contexts was 3,238. Both CPU and GPU implementations were compiled by the NVIDIA CUDA compiler version 3.0 (64-bit) to avoid any compiler effects on the experimental results. Optimization (`-O3 -msse4.2`) and OpenMP-based parallelization[4] (`-fopenmp -lgomp`) were both used.

Table 2 shows the experimental results. It can be seen from the table that the proposed implementation running on a GPU was about 9.7 times faster than the HTS implementation running on CPUs. There was common overheads between these two implementations, such as loading and saving models and building the bitfield which represents matching between model name and questions. If these common overheads were excluded, the decision tree-based context clustering procedure itself in the proposed implementation was about **11.2** times faster than that of the HTS implementation.

Table 2: *Average computational time for building 10 decision trees.*

Architecture	Computational time (sec.)
CPU (HTS)	47,637
GPU (Proposed)	4,903
(common overheads)	710

The HTS implementation of decision tree-based context clustering performs sophisticated question pruning, *e.g.*, do not evaluate questions which split clusters 100% vs. 0%, to reduce the computational cost. However, this pruning mechanism has not been integrated into the GPU implementation. The GPU implementation can be expected to become even faster if this pruning mechanism can be integrated.

6. Conclusions

The implementation of decision tree-based context clustering for a parallel architecture was given. The performance results were pleasing and show a considerable improvement over sequential implementation. A number of limitations were encountered and overcome. In addition other improvements have been identified that could be implemented in the future to allow for better and more robust performance.

7. Acknowledgements

This work was performed while the first author was an intern in the Speech Technology Group, Toshiba Research Europe Ltd. Cambridge Research Laboratory in summer 2009.

8. References

[1] T. Yoshimura, K. Tokuda, T. Masuko, T. Kobayashi, and T. Kitamura, "Simultaneous modeling of spectrum, pitch and duration in HMM-based speech synthesis," in *Proc. Eurospeech*, 1999, pp. 2347–2350.

[2] J. Odell, "The use of context in large vocabulary speech recognition," Ph.D. dissertation, Cambridge University, 1995.

[3] K. Tokuda, K. Oura, K. Hashimoto, H. Zen, J. Yamagishi, T. Toda, T. Nose, S. Sako, and A. Black, "The HMM-based speech synthesis system (HTS)," 2010, http://hts.sp.nitech.ac.jp/.

[4] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X.-Y. Liu, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. Woodland, "The hidden Markov model toolkit (HTK) version 3.4.1," 2009, http://htk.eng.cam.ac.uk/.

[5] *NVIDIA CUDA Programming Guide*, 2009.

[6] P. Dixon, T. Oonishi, and S. Furui, "Harnessing graphics processors for the fast computation of acoustic likelihoods in speech recognition," *Comput. Speech Lang.*, vol. 23, no. 4, pp. 510–526, 2009.

[7] Y. Shi, F. Seide, and F. Soong, "GPU-accelerated Gaussian clustering for fMPE discriminative training," in *Proc. Interspeech*, 2008, pp. 944–947.

[8] K. Shinoda and T. Watanabe, "Acoustic modeling based on the MDL criterion for speech recognition," in *Proc. Eurospeech*, 1997, pp. 99–102.

[9] H. Zen, T. Toda, M. Nakamura, and T. Tokuda, "Details of the Nitech HMM-based speech synthesis system for the Blizzard Challenge 2005," *IEICE Trans. Inf. Syst.*, vol. E90-D, no. 1, pp. 325–333, 2007.

[3] Five trees for the stream modeling spectral parameters and another five trees for the stream modeling $\log F0$ parameters were built in serial by a single `HHEd` process.

[4] The latest HTS release, HTS-2.1.1β, supports OpenMP-based parallelization while creating the bitfield.