Rapid Training of Acoustic Models using Graphics Processing Units

Senaka Buthpitiya, Ian Lane, Jike Chong

Department of Electrical and Computer Engineering, Carnegie Mellon University

{senaka.buthpitiya, ianlane, jike.chong}@sv.cmu.edu

Abstract

Robust and accurate speech recognition systems can only be realized with adequately trained acoustic models. For common languages, state-of-the-art systems are now trained on thousands of hours of speech data. With even a large cluster of machines the entire training process can take many weeks. To overcome this development bottleneck we propose a new framework for rapid training of acoustic models using highly parallel graphics processing units (GPUs). In this paper we focus on Viterbi training and describe the optimizations required for effective throughput on GPU processors. Using a single NVIDIA GTX580 GPU our proposed approach is shown to be 51x faster than a sequential CPU implementation, enabling a moderately sized acoustic model to be trained on 1000 hours of speech data in just over 9 hours. Moreover, we show that our implementation on a two-GPU system can perform 67% faster than a standard parallel reference implementation on a high-end 32-core Xeon server. Our GPU-based training platform empowers research groups to rapidly evaluate new ideas and build accurate and robust acoustic models on very large training corpora.

Index Terms: Continuous Speech Recognition, Acoustic Model Training, Graphics Processing Unit

1. Introduction

The availability of very large training corpora (1000 hours and more) are empowering speech researchers to achieve ever higher accuracy on challenging speech recognition tasks. However, training of acoustic models on these large corpora can take weeks, even on large clusters of workstations. This limits the number of new ideas and concepts that can be explored and validated in a timely manner. In this paper we introduce a novel approach to rapidly train acoustic models using affordable ($500) off-the-shelf graphics processing units (GPUs). Our platform can train an acoustic model at 220x the speed of an equivalent implementation on a CPU (without pruning) and 51x faster than standard Viterbi-based training with pruning. This platform is ideal for accelerating the exploration and validation of ideas for automatic speech recognition.

A common method for training Hidden Markov Model (HMM)-based acoustic models is Viterbi Training. Figure 1 illustrates the main steps of this process, which include (0) loading training data, (1) computing observation probabilities, (2) assigning observations to specific states and Gaussian components within the model (E-Step), and (3) collecting statistics from the expectation step to reestimate model parameters (M-step).

There has been a number of efforts over the past decades to reduce the time required to train acoustic models for speech recognition. In 1990, Pepper et al. experimented with performing training on a set of computers organized in a ring [2]. In 1992, Foote et al. introduced an approach to distribute HMM training to a set of five loosely-coupled Armstrong II multiprocessor network computers. In 1997, Yun et al. mapped the training algorithm to an FPGA infrastructure [3] and in 2006 Popescu et al. implemented acoustic model training on a MPI-based cluster with three nodes [4]. These prior works all achieved less than 3x speedup over sequential runs and thus have not been widely used.

The availability of general-purpose programmable GPU and data parallel programming models [5] has opened up new opportunities to train speech models at orders of magnitude faster than before. This is further empowered by new algorithms and implementation techniques that focus on parallel scalability [6], which expose the fine-grained concurrency in compute-intensive applications and exploits the concurrency on highly parallel manycore microprocessors.

In cuHMM [7], Liu implemented training of discrete HMMs on GPUs. This generic training engine, although effective for applications such as biological sequence analysis, is not appropriate for acoustic model training as it is unable to handle continuous observation models and cannot take advantage of the special left-right model structure used in speech recognition. In [8], Dixon et al. introduced techniques for fast acoustic likelihood computation in the context of a speech recognition decoder, but did not extend the work to the training process and in [9] Pangborn constructed an efficient implementation on the GPU for flow cytometry used in biology and immunology. This approach, however, only trained a single Gaussian mixture model (GMM) and is thus unsuitable for acoustic model training. In this paper we describe an optimized infrastructure for training HMMs, where we leverage the special left-right HMM model structure commonly used in speech recognition while heavily optimizing the observation probability computation.

As the development platform we use the NVIDIA GTX580 GPU which contains 16 cores on-a-chip, two 16-wide SIMD pipelines in a core, as well as hardware managed cache and software managed memory scratch pad. The GPU is programmed using CUDA [5], a representative data-parallel manycore programming language where an application is organized into a se-

Figure 1: Training flow for one training iteration

(0) Load Batch of Utterances
(1) Compute Observation Probabilities
(2a) Viterbi Alignment E-Step
(2b) Back Trace Best Path
(3a) Accumulate $\mu$, $\nu$ and Transition Counts
(3b) Generate New Global Model
(4) Complete All Batches?
Yes
No

M-Step
parallel sequential program that will be mapped across a set of parallel threads, which are organized into groups called thread blocks.

The challenge is to effectively organize the training algorithm into threads and thread-blocks and leverage available memory resources and synchronization capabilities to efficiently execute on a manycore computation platform.

### 2. Viterbi Training of Acoustic Models

Viterbi training is a common method for maximum likelihood re-estimate of parameters of an acoustic model. Given a set of training observations $O^r$, $1 \leq r \leq R$ and an HMM state sequence $1 < j < N$ the observations sequence is aligned to the state sequence via Viterbi alignment. This alignment results from maximizing

$$\phi_X(T) = \max_i [\phi_i(T) a_i N] \quad (1)$$

for $1 < i < N$ where

$$\phi_i(t) = b_j(o_t) \max \left\{ \phi_{j-1}(t-1) a_{ij} \right\} \quad (2)$$

with initial conditions, $\phi_1(1) = 1$ and $\phi_N(1) = a_1 b_j(o_t)$, for $1 < j < N$. When observation likelihoods are modeled as mixture Gaussian densities the output probability $b_j(o_t)$ is as defined as:

$$b_j(o_t) = \sum_{m=1}^{M_j} c_{jm} N(o_t; \mu_{jm}, \Sigma_{jm}) \quad (3)$$

where $M_j$ is the number of mixture components in state $j$, $c_{jm}$ is the weight of the $m^{th}$ component and $N(\cdot ; \mu, \Sigma)$ is a multivariate Gaussian with mean vector $\mu$ and covariance $\Sigma$. In Viterbi training, model parameters are updated based on the single-best alignment of individual observations to states and Gaussian components within states. From this alignment, transition probabilities are estimated from the relative frequencies

$$\hat{a}_{ij} = \frac{A_{ij}}{\sum_{k=2}^{N} A_{ik}} \quad (4)$$

where $A_{ij}$ it the total number of transitions from state $i$ to state $j$. The means and variances of the observation densities are updated using an indicator function $\psi^r_j(t)$ which is 1 if $o^r_t$ is associated with mixture component $m$ of state $j$ and is zero otherwise.

$$\hat{\mu}_{jm} = \frac{\sum_{r=1}^{R} \sum_{t=1}^{T_r} \psi^r_j(t) o^r_t}{\sum_{r=1}^{R} \sum_{t=1}^{T_r} \psi^r_j(t)} \quad (5)$$

$$\hat{\Sigma}_{jm} = \frac{\sum_{r=1}^{R} \sum_{t=1}^{T_r} \psi^r_j(t) (o^r_t - \hat{\mu}_{jm}) (o^r_t - \hat{\mu}_{jm})^T}{\sum_{r=1}^{R} \sum_{t=1}^{T_r} \psi^r_j(t)} \quad (6)$$

and the mixture weights are computed based on the number of observations allocated to each component

$$c_{jm} = \frac{\sum_{r=1}^{R} \sum_{t=1}^{T_r} \psi^r_j(t)}{\sum_{r=1}^{R} \sum_{t=1}^{T_r} \sum_{M} \psi^r_j(t)} \quad (7)$$

Viterbi training provides a fast method to perform maximum likelihood re-estimation of acoustic models and in most cases is just as effective as the Baum-Welch method [10].

### 3. Viterbi Training on Manycore Processors

Training is a highly data-parallel operation involving the aggregation of statistics from a large training data set possibly containing millions of utterances. Concurrency exists both between utterances and within an utterance, making the training process highly amenable for parallelization. However, constructing an efficient parallel implementation requires not only extensive application concurrency, but also a deep understanding of the available parallel computation resources.

#### 3.1. Step 1: Observation Probability Computation

The observation probability computation step implements Equation 3, and contains five levels of concurrency: among features, among mixture components, among GMM, among input observations, and among utterances, as illustrated in Table 1.

<table>
<thead>
<tr>
<th>Concurrency</th>
<th>Parallelism</th>
<th>Task Size (# IPT)</th>
<th>Data Size (# values)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features</td>
<td>39</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Mixtures</td>
<td>32</td>
<td>100</td>
<td>80</td>
</tr>
<tr>
<td>GMMs</td>
<td>100</td>
<td>4000</td>
<td>2500</td>
</tr>
<tr>
<td>Observation</td>
<td>360</td>
<td>400K</td>
<td>250K</td>
</tr>
<tr>
<td>Utterance</td>
<td>1.1M</td>
<td>144M</td>
<td>250K</td>
</tr>
</tbody>
</table>

**Parallelism:** we use a model with 39 dimensional features, 32 mixture components per GMM, and a training set consisting of, on average, 3.6 seconds long audio segments, transcripts of 5-10 words with 100 GMMs per utterance, and a total of 1.1M utterances representing a conversational speech training corpus of 1000 hours of audio.

**Task Size:** number of instructions that can execute in a thread before a synchronization event or task completion. For example, when a feature is assigned to a thread, each thread can only perform a “weighted-difference” calculation (in two instructions) before synchronizing to sum the “weighted-differences” between threads; for mixture-level parallelism, each thread sums the “weighted-differences” sequentially, and synchronize to calculate the weighted-sum of mixtures.

**Data Size:** size of the data working set required to perform the task within each thread. For feature-level parallelism, each thread only require the mean and variance of a feature dimension. For mixture level parallelism, each thread requires 39 mean, 39 variance, as well as a likelihood constant and weight of the mixture, for a total of 80 values for the computation. For GMM-level computation, each thread requires access to all 32 mixtures for a total of 2560 values for the computation.

For efficient implementation on the GPU, we choose the level of parallelism to maximize task granularity while allow the data working size to be small enough to fit into fast hardware-managed cache and software-managed memory scratch space. In this case, it is GMM-level parallels with a data size of 2560 values represent 10KB of model data, which fits into the shared memory scratch space on the GPU. We then organize the threads to parallelize over the observation samples, and the thread blocks to parallelize over the GMMs. This maximizes the model data reuse from the on-chip shared memory and pre-computing the observation probabilities to construct a codebook for the next steps. The input observation data for each utterance is transposed such that there is one array for each feature dimension over time steps. This allows each thread in a thread block to work on all computations for one time step.

#### 3.2. Step 2a: Alpha Computation

The Alpha computation is the forward pass of the Viterbi algorithm that uses dynamic programming to arrive at an optimal estimation of match likelihood between the transcript and the
acoustic input (Equation 2). It is implemented as a time syn- 
chronous operation where the computation for one acoustic in- 
put sample is dependent on the results from the previous input 
sample. We parallelize the computation by mapping utterances 
over thread blocks. Each thread block then iterates sequentially 
over the time steps, with each thread handling the computation 
for one GMM state. This implementation allows model transi- 
tion probabilities and partial results to be cached in shared 
memory, enabling fast iteration of computation without incurr- 
ing excessive memory operations to off-chip memory.

3.3. Step 2b: Backtracking Computation

The backtrack computation traces the one-best path that demon- 
strates the best alignment of GMM states to acoustic input ob- 
servations. It starts from the end of the GMM state sequence 
representing the transcript and the end of the acoustic input 
sequence, and backtracks step by step to the beginning of the 
GMM state sequence. Implemented naively, this is a severe 
performance bottleneck. The pointer chasing operation will in- 
volve two memory round trips per time step, which can taking 
100,000s processor cycles to backtrack 400 steps.

We implemented this step use a prefetch optimization. By 
prefetch 32 time steps of potentially accessible values as a batch 
into the shared memory on the GPU and perform the backtrack 
from shared memory, we fully utilize the load bandwidth and 
minimized memory latency caused by the pointer chasing oper- 
ations in the backtrack process.

3.4. Step 3: Maximization Step

The maximization step takes the aligned and labeled input ob- 
servations to update the aggregated statistics in the acoustic 
model according to equations 4, 5, 6 and 7. The process is 
an instance of histogram generation, where we compute the sta- 
tistical distribution of the training data set. We use a hybrid 
local-global accumulation method to efficiently aggregate the 
statistics in the histogram generation process. A typical train- 
ing data set for speech recognition set may involve thousands 
of hours of audio, stored as segments of audio that represents 
speech utterances separated by silences, taking tens to hundreds 
of gigabytes of storage.

The histogram bins are the Gaussian mixtures in the GMMs 
representing triphone states in the acoustic speech model. An 
acoustic speech model typically contains 10,000 triphone states 
in an acoustic codebook. For a 32 mixture Gaussian model, 
there can be as many as 320,000 histogram bins to accumulate to, 
each represented by a 39-dimensional mean and variance 
value-pair. This number of bins would be too large to fit in the 
last level cache of today’s microprocessors, making an efficient 
histogram generation implementation challenging.

Given the training data set and histogram characteristic, we 
map each utterance to a thread block on the GPU, and first ag- 
gregate the histogram information within an utterance locally, 
then merging the results from each thread block to the main his- 
togram globally. While this does not solve the uncacheably 
large number of histogram bins, it does alleviate the potential 
sequentialization bottlenecks at some histogram bins when 
thousands of thread context concurrently performing memory 
operations on a popular triphone state. For the aggregation of 
floating-point values, our implementation extensively lever- 
age floating-point atomic additions to global memory in the 
NVIDIA Fermi architecture.

4. Results

4.1. Experimental Evaluation

We evaluated the effectiveness of our proposed GPU-based 
acoustic model training procedure by first comparing the time 
required to perform one iteration of Viterbi training to a single-
thread CPU implementation. The speech corpora used in this 
evaluation consisted of 122hrs and approximately 150k utter- 
ces of speech collected from headset, lapel and far-field mi- 

crophones from 168 sessions from the AMI Meeting Corpus.

This data was replicated to generate the larger training sets up 
to 10,000 hours. An initial cross-word, context dependent triphone 
model was trained on this corpora using HTK [11] and the re- 
sulting model consisted of 8000 codebooks with a maximum of 
32 Gaussian components per codebook. Each model consisted 
of a three-state left-to-right hidden Markov model (HMM) with 
two transitions per state, a local transition to the current state 
and a transition to the neighboring state to the right. Models 
were trained using 39-dimension acoustic features, which com- 
bined 13-dimension MFCCs with its delta and delta-delta com- 
ponents.

For evaluation we used the NVIDIA GTX580 GPU in an 
Intel Core i7-2600k CPU-based host platform. The GPU has 16 
cores each with dual issue 16-way vector arithmetic units run- 
ning at 1.54GHz. Its processor architecture allows a theoretical 
maximum of two single-precision floating point operations (SP 
FLOP) per cycle, resulting in a maximum of 1.58 TeraFLOP 
of peak performance per second. The GPU device has 1.5GB 
of GDDR5 memory. For CPU runs, we used a 4-socket Intel 
Xeon X7550 Server running at 2.00GHz with an aggregate of 
32 cores and 128GB of memory. For compilation, we used g++ 
4.5.0 and NVCC 3.2 targeting GPU Compute Capability 2.0.

4.2. Performance Analysis

Figure 2 illustrates the training time used for various training set 

sizes. As expected, training time scales linearly with increase 
in training set size. The CPU run-times were measured with 
and without pruning using Viterbi training. Using one hour of 
training time, we can process 7.2 hours of training data with 
no pruning, and 31.3 hours of data with pruning. With GPU-based 
implementation, we can process 1582.3 hours of data with no 
pruning using an equivalent manycore Viterbi training routine. 
That is a 220x faster than the same algorithm and 50.6x faster 
than a more advanced algorithm with pruning running sequen- 
tially on the Xeon server CPU. Given a 10,000-hour training 
corpus, a single GPU is able to perform one iteration of viterbi 
training in just 6.3 hours this compares with 1400 hours, for 
the CPU implementation without pruning and 320 hours for the
One approach to train a full acoustic model involves iteratively increasing the number of Gaussian components from a single Gaussian to a much larger number via mixture splitting. This process is computationally expensive and involves performing 3-5 EM training iterations for acoustic models with increasingly large observation models (GMMs). For example, to train a 32-component GMM the training time required on a single GPU for a 1000-hour training set is illustrated in Table 2. We assume an average of four EM iterations are used per mixture count, and accumulate the measured execution time for our Viterbi training routine on the GPU. Taking into account the time necessary for performing mixture splitting, we expect to be able to train an acoustic model with 1000-hour of data over night in less than 9.25 hours, with 0.37 hours accounted for mixture splitting and cluster tying overhead. Our rapid training system allows new ideas and concepts to be tested on a 100-hour training set in less than one hour.

Table 2: Training time for a 1000-hour training set for models with different numbers of Gaussian mixtures

<table>
<thead>
<tr>
<th># components in GMM</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>One iteration (hours)</td>
<td>0.26</td>
<td>0.27</td>
<td>0.29</td>
<td>0.34</td>
<td>0.43</td>
</tr>
<tr>
<td>Four iterations (hours)</td>
<td>1.02</td>
<td>1.06</td>
<td>1.17</td>
<td>1.36</td>
<td>1.74</td>
</tr>
<tr>
<td>Total (hours)</td>
<td>9.25 + 8.88 + 0.37 (overhead)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3 illustrates the training time achievable with a 4-sOCKET 32-core Xeon server costing over $30,000. Given a 1000-hour training set, the utterances are dynamically scheduled onto a varying number of CPU threads. We see that the 32-core Xeon server only has 7.5% performance advantage compared to a single GPU system, and that a two-GTX580 desktop system that cost less than $2,000 can be 67% faster than a 32-core Xeon server for performing one EM iteration with 32-component GMMs on a 1000-hour training set.

Figure 4 illustrates the step-wise timing break down for 1000-hour training set. The runs are separated into batches to allow the data working set within a batch to fit on the GPU global memory. The most compute-intensive observation probability computation uses 68.4% of the total execution time. We have achieved 1.23 instructions per cycle (IPC) on the Fermi GPU architecture, which is 61.5% of the peak execution efficiency. The E-Step includes the highly sequential backtracking process, with the various technique described in section 3.2 and 3.3, only 12.9% of the time is spent here. By using the local-global reduction technique and leveraging efficient floating-point atomic-add capability supported by the hardware, the M-Step takes only 6.6% of total execution time. The utterance load step is taking up quite significant 12% of the total executing time. This step is expected to be significantly faster after an re-factoring the code base and is on the list of future work.

4.3. Discussion

The AMI Meeting Corpus consists of relatively short conversational utterances that are on average only 3.6 seconds long. 69% of the utterances are less than 2 seconds long. With short utterances, performing force-alignment without pruning incurs little overhead. We are working on implementing some pruning techniques for corpora with longer average utterance length.

5. Conclusion

We presented a new framework for rapid training of acoustic models using the GPU. We focus on Viterbi training and shown that using a single GPU, our proposed approach is 51x faster than a sequential CPU implementation. Training an acoustic model with 8000 codebook of 32-component Gaussian mixtures on 1000 hours of speech would take just over 9 hours. Our GPU-based training platform empowers research groups to rapidly evaluate new ideas and build accurate and robust acoustic models on very large training corpora. In future work we intend to also investigate Baum-Welch training and MMIE-based discriminative training on this platform.

6. References