MULTI-USER REAL-TIME SPEECH RECOGNITION WITH A GPU

Jungsuk Kim and Wonyong Sung

Department of Electrical Engineering and Computer Science, Seoul National University
599 Gwanangno, Gwanak-gu, Seoul, 151-744, Korea
Email: kimjs@dsp.snu.ac.kr, wysung@snu.ac.kr

ABSTRACT

We have developed a multi-user large vocabulary speech recognition system employing a fully composed one-level weighted finite state transducer (WFST) based network on a Graphics Processing Unit (GPU). This system improves the overall throughput and latency of speech recognition engine which processes multiple users' utterances at the same time with efficient scheduling, parameter sharing, and communication overhead reduction techniques. We conduct both batch speech simulation and trace driven online simulation to access the performance of the developed system. Traces are generated based on a queueing model.

Index Terms— Speech recognition, LVCSR, GPU, Distributed Speech Recognition, WFST

1. INTRODUCTION

Automatic speech recognition (ASR) is widely adopted as a convenient user interface in many handheld devices. Since large vocabulary speech recognition on an embedded system, such as a cellular phone, consumes much power, it is desired to employ the distributed speech recognition (DSR) technology that conducts speech recognition on a remote server [1]

The principal client-server framework of the DSR is presented in Fig. 1. The client side performs the front-end of the speech recognition system such as the feature parameter extraction. These features are transmitted over a communication channel, and delivered to remote back-end servers. In the server side, multiple requests are dispatched by the speech recognition proxy (SRP) and processed by speech recognition engines [1]. In practice, a DSR server should quickly answer to multiple requests simultaneously with given computing resources. Most of the the current speech enabled web servers apply single user speech recognition in a time-multiplexed way. This system can be modeled as an $M/M/1$ queue model

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2011-0027502) and by the MKE (The Ministry of Knowledge Economy), Korea and Microsoft Research, under IT/SW Creative research program supervised by the NIPA (National IT Industry Promotion Agency) (NIPA-2011-C1810-1102-0018), also by the Brain Korea 21 Project.

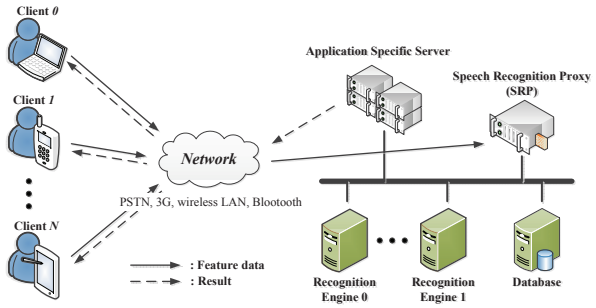


Fig. 1. The principal client-server framework of the Distributed Speech Recognition.

which consists of a single wait queue and a single server [2]. Note that the notation $M/M/1$ is used to describe a queueing system, where the first M specifies that the arrival process is Poisson, second M denotes that the service times are i.i.d. exponential random variables, 1 specifies the number of servers. In this model, the server only processes one request at a time, thus the input utterances from multiple users have to wait in the queue until the server is available. Therefore, the wait time in the queue can increase the latency and reduce the throughput, especially in heavy work load situations.

Recently graphic processing units (GPUs) gain popularity not just in personal desktop but also in server systems. Because the speech recognition process embeds many fine-grained parallelism, there have been various researches on the parallelization of them on a GPU [3, 4, 5, 6, 7]. The GPU based speech recognition engine is usually much faster than real-time for a single user service. However, for a multi-user service, the wait time minimization is greatly needed. But, previous approaches concentrate on improving the recognition speed of a single utterance. In this work, we try to reduce the latency and increase the throughput of multi-user speech recognition with a GPU based server, instead of just minimizing the execution time for a signal user speech.

This paper is organized as follows. Section 2 presents optimization techniques including efficient scheduling, parameter sharing and communication overhead reduction. Section 3 shows the experimental results. Finally, concluding remarks are given in Section 4.

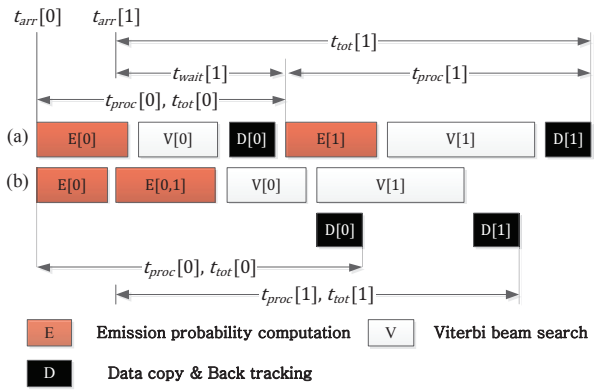


Fig. 2. Scheduling schemes of multiple utterances. (a) Time multiplexed (b) Proposed

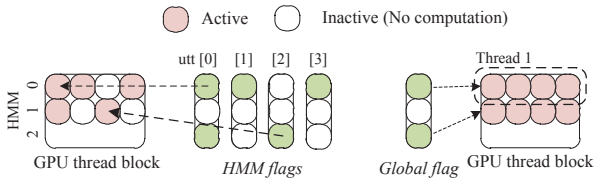
2. OPTIMIZATION TECHNIQUES FOR MULTI-USER SPEECH RECOGNITION

2.1. Efficient scheduling of multiple utterances

The WFST based speech recognition algorithm consists of two major phases; phase 1 computes emission probability and phase 2 conducts Viterbi beam search over the WFST network [4]. In GPU based implementations, phase 3 is needed for moving intermediate search results between the GPU and the CPU.

A conventional single user time-multiplexed scheduling processes multiple utterances sequentially as shown in Fig. 2(a). The first input utterance $utt[0]$ arrives the queue at $t_{req}[0]$ and the recognition engine starts to process it immediately. Note that the request takes the processing time of $t_{proc}[0]$ to complete the recognition process, and the processing time with a GPU is much shorter than the speech duration of the utterance. While the recognition engine is occupied by $utt[0]$, the other utterances such as $utt[1]$ in the queue have to wait $t_{wait}[1]$ as shown in Fig. 2(a). The total processing time $t_{tot}[n]$ of the request $utt[n]$ is the sum of the wait time $t_{wait}[n]$ and the processing time $t_{proc}[n]$.

The proposed scheduling conducts the multiple speech recognition simultaneously as shown in Fig. 2(b). We try to simultaneously conduct the emission probability computation of multiple utterances that are at the waiting queue. Thus, $utt[1]$ can be processed as soon as it arrives. In Fig. 2(b), E[0,1] represents the period that computes the emission probability of $utt[0]$ and $utt[1]$ concurrently. After phase 1 is finished, the Viterbi search of each utterance is conducted in a time-multiplexed way because the Viterbi search demands large overheads in context switching between the utterances and also consumes a large size of memory. We adopt a double buffering technique and page-locked memory to minimize the communication overhead by conducting phase 3 at the background illustrated in Fig. 2(b). Hence, $t_{tot}[0]$ increases but, $t_{tot}[1]$ decreases and the overall throughput improves much.



(a) Distributed flag buffer (b) Global flag buffer

Fig. 3. Beam pruning strategies of the emission probability computation.

As a result, the request in the waiting queue can be processed with short delay than conventional speech recognizers. In the following section, the optimization techniques are presented with respect to each recognition phase.

2.2. Gaussian parameter sharing in the emission probability computation

The emission probability computation is the most time consuming phase in the LVCSR system, which takes 30-70 percent of the total recognition time [6]. This phase computes the likelihood for all pairs of feature vectors and Gaussians in the Gaussian mixture models (GMMs) that are shared among the HMM states.

For each feature vector $\mathbf{o}_t$ of dimension D we calculate the likelihood of each Gaussian j, $1 \leq j \leq M$ in the GMM k. The total number of GMM is G and these GMMs are used by the system to model various states of HMMs in the speech recognition model. If the covariance matrix of Gaussian j is assumed diagonal, the likelihood computation between GMM k and feature vector $\mathbf{o}_t$ can be expressed as:

$$b_k(\mathbf{o}_t) = \sum_{j=1}^{M} c_{kj} \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi\sigma_{kjd}^2}} \exp\left(-\frac{1}{2}\frac{(o_{td}-\mu_{kjd})^2}{\sigma_{kjd}^2}\right), \quad (1)$$

where μ_{kjd}, σ_{kjd} and c_{kj} are the mean, variance and weight of the Gaussian j.

In Eq.(1), the GMM parameters c_{kj}, μ_{kjd} and σ_{kjd} are loaded to the shared memory on the GPU at the beginning of the emission probability computation phase to maximize data re-useability [7]. In the multi-user case, we gather active HMM states that share the same GMMs and allocate them in the same GPU threads block. Therefore every HMM state in the same GPU threads block can share these Gaussian parameters and minimize the number of global DRAM accesses.

Under the beam pruning strategy, we only need to calculate the emission probabilities of active HMM states that are inside the beam [3, 6, 7]. To indicate activeness of HMM states, we set flags of the buffer when the HMM state is inside the beam. This strategy can be extended to the concurrent N user case easily by adding extra flag buffers for each utterances as shown in Fig. 3(a).

This distributed flag method demands minimum computation, because it skips out all of the inactive HMM states. However, this method needs N times more conditional operations to know the activeness, and it also needs N times more memory space to save the flags. Note that N is the number of users serviced at the same recognition server. Moreover, the conditional branches incur warp divergence that can lead to significant loss of parallel efficiency [8].

We merge the flag buffers into one global flag buffer as shown in Fig. 3(b). Every active HMM state in different utterances uses the global flag buffer if those active HMM states share the same GMM. Therefore some emission probabilities of inactive HMM states are calculated if the HMM states are active in other utterances. The global flag buffer method needs some redundant computation but greatly improves the parallel efficiency by reducing the warp divergence.

2.3. Communication overhead minimization in the Viterbi beam search

During the Viterbi beam search, the recognition engine needs to save intermediate search results, such as the list of active states and the accumulated negative log likelihood of each state, in the buffer [3, 7]. Therefore, we need an N times bigger buffer to evaluate N utterances concurrently. However the GPU has a relatively limited DRAM space, hence N should be selected carefully considering the size of memory footprint in the GPU. To minimize the memory footprint, we add an extra buffer in the host platform (CPU) and move the intermediate search result between the CPU and the GPU. However, the communication between the GPU and the CPU is not fast enough [8].

To minimize the communication overhead between the CPU and the GPU, a double buffering technique is adopted using the page-locked memory. The mapped page-locked host memory guarantees that the operating system will never page out this memory to the hard disk, which ensures its residency in the physical memory. Knowing the physical address of the buffer, the GPU can use the direct memory access (DMA) to copy data to or from the host very efficiently. Also, the data transfer operations between the page-locked memory and the device memory can be performed concurrently with the GPU kernel execution [8]. For example, we can move the intermediate search data of $utt[1]$ to the buffer on the GPU from the CPU when $utt[0]$ is searched with the other buffer. Using the mapped page-locked memory and the double buffering technique, we can hide the communication time that exchanges the intermediate search data.

3. EXPERIMENTAL RESULTS

3.1. Experiment setup

We use an NVIDIA GTX 580 (Fermi architecture) GPU which has 30 CUDA cores with an Intel Core i7 980 based host platform [8].

Table 1. Performance summary of the batch speech transcript simulation.

	Baseline	Distributed	Global
N(# of concurrent processes)	1	3	3
Phase 1 (ms)	17,075.0	24,108.8	14,637.0
Phase 2 (ms)	63,696.5	63,871.3	63,921.9
Phase 3 (ms)	3,337.4	0.0	0.0
Total (s)	84.87	88.61	79.19
Speed up (Phase 1)	1.00×	0.71×	1.17×
Speed up (Total time)	1.00×	0.95×	1.07×

The acoustic model was trained by HTK, with the Wall Street Journal(WSJ) 1 corpus and the test was conducted with WSJ1 5k test and evaluation set which has 330 test utterances. Each utterance has an average duration of 7.3 sec. The trained acoustic model contains 3,000 16-mixture Gaussians and 39 dimensional MFCCs are used as feature vectors.

The fully composed WFST network is compiled and optimized offline [9]. The network is modified to one-level step by adopting look-overhead epsilon arcs and transforming epsilon arcs to non-epsilon arcs [4]. The resultant WFST network consists of 3.9 million states and 14.5 millon non-epsilon arcs.

The word error rate (WER) is set to 9.53% by adjusting the search beam width dynamically depending on the number of active arcs. The baseline GPU-based speech recognition engine was implemented using the efficient data packing method as explained in [7]. We use the real time factor (RTF) as a performance measure, which is computed as the total decoding time divided by the input speech duration. Note that the total decoding time is the the sum of the wait time and the processing time.

3.2. Performance of the batch speech transcription simulation

As shown in Table 1, the emission probability computation using the global flag buffer achieved 16.7 % of speed-up improvement when N is 3. The distributed flag buffer method that incurred the warp divergence increased the total recognition time by 4.4% and the emission probability computation time by 41.1%. Although the global buffer method demands more computation, the execution time is much reduced because of improved parallel efficiency. Also, 4.8% of time is reduced in the phase 2, 3 by overlapping data transfer and GPU kernel execution time.

3.3. Performance of the trace driven on-line simulation

The request arrivals are modeled as a Poisson arrival process with the average rate λ per second. We model this by generating a set of arrival times for each request from a user. We

Table 2. Performance summary of the trace driven online simulation.

	N	λ (Average Request Rate per second)						
		1.0	1.5	2.0	2.5	3.0	3.5	4.0
t_{wait}(s;%)	1	42.23(13.7)	85.81(24.4)	121.15(31.3)	281.34(51.4)	434.46(62.0)	1,277.29(82.8)	2,858.76(91.5)
	2	33.79(10.9)	62.16(17.4)	81.72(20.9)	153.16(30.2)	278.3(43.6)	657.74(60.5)	1,113.26(71.7)
	3	33.58(10.8)	57.10(15.8)	70.34(17.7)	121.56(23.3)	233.65(35.7)	522.61(47.9)	884.67(59.9)
t_{proc}(s;%)	1	265.72(86.3)	266.09(75.6)	266.16(68.7)	265.90(48.6)	266.17(38.0)	265.85(17.2)	265.95(8.5)
	2	244.08(89.1)	295.89(82.6)	308.59(79.1)	353.34(69.8)	360.15(56.4)	428.48(39.4)	438.51(28.3)
	3	277.33(89.2)	303.45(84.2)	328.11(82.4)	399.95(76.7)	419.62(64.3)	568.17(52.1)	591.76(40.1)
Avg. RTF	1	0.04	0.05	0.06	0.08	0.10	0.24	0.50
	2	0.04	0.05	0.06	0.07	0.09	0.16	0.24
	3	0.04	0.05	0.06	0.07	0.09	0.16	0.23
Max RTF	1	0.14	0.20	0.27	0.50	1.01	1.20	2.38
	2	0.11	0.17	0.22	0.38	0.78	0.86	1.46
	3	0.11	0.17	0.22	0.34	0.71	0.83	1.29

generate a pseudo random number r uniformly distributed in the interval $[0, 1]$ and then compute the arrival time of nth request $t_{arr}[n]$ as follows:

$$t_{arr}[n] = t_{arr}[n-1] - \frac{1}{\lambda}\log(r). \quad (2)$$

We generate 10 traces for each average rate that increases from 1.0 to 7.5 with the step of 0.5. Each trace has 330 user requests from the test and evaluation set of WSJ1 corpus.

The maximum and average RTF indicate the worst and the average case recognition speeds respectively. A system does not meet the real-time performance when the maximum RTF is bigger than 1 even if the average RTF is smaller than 1. As shown in Table 2, the maximum RTF of the baseline (N=1) is 1.01 when λ is 3.0 but the maximum RTF decreases noticeably when N is 2 or 3. Therefore the developed multi-user speech recognizers show much better worst case performance when compared with the single user speech recognition engine. When λ is 3.5 and N is 3, the recognizer is 1.49 time faster in the average and 1.45 times faster in the worst-case compared with the baseline.

4. CONCLUDING REMARKS

We have implemented a multi-user speech recognition engine using an NVIDIA GTX 580 GPU employing a fully composed one-level WFST network. The developed system processes multiple users' utterances simultaneously to reduce the wait time. We achieve 1.67× speed up in the emission probability computation by sharing Gaussian parameters among utterances using the global flag buffer. Also, we adopt a double buffering technique with mapped page-locked memory to minimize the communication overhead by overlapping the data transfer and GPU kernel execution time. As a result, we obtained 1.49× in the average and 1.45× in the worst case speed-up when compared with a system based on a time-multiplexed single user speech recognition engine on a GPU.

5. REFERENCES

[1] W. Zhang, L. He, Y.-L. Chow, R. Yang, and Y. Su, "The study on distributed speech recognition system," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2000, pp. 1431–1434.

[2] D. Gross and C. Harris, *Fundamentals of queueing theory*, Wiley Series in Probability and Statistics, 4th edition, 2008.

[3] K. You, J. Chong, Y. Yi, E. Gonina, C. J. Hughes, Y.-K. Chen, W. Sung, and K. Keutzer, "Parallel scalability in speech recognition," *IEEE Signal Processing Magazine*, vol. 26, no. 6, pp. 124–135, Nov. 2009.

[4] J. Chong, E. Gonina, Y. Yi, and K. Keutzer, "A fully data parallel WFST-based large vocabulary continuous speech recognition on a graphics processing unit," in *Proc. Interspeech*, Sep. 2009, pp. 1183–1186.

[5] P. R. Dixon, T. Oonishi, and S. Furui, "Fast acoustic computations using graphics processors," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Apr. 2009.

[6] P. Květoň and M. Novák, "Accelerating hierarchical acoustic likelihood computation on graphics processors," in *Proc. Interspeech*, Sep. 2010.

[7] J. Kim, K. You, and W. Sung, "H- and C-level WFST-based large vocabulary continuous speech recognition on graphics processing units," in *IEEE Internation Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2011, pp. 1733–1736.

[8] NVIDIA, *NVIDIA CUDA Programming Guide Version 4.0*, May 2011.

[9] 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.