GPU-ACCELERATED VITERBI EXACT LATTICE DECODER FOR BATCHED ONLINE AND OFFLINE SPEECH RECOGNITION

Hugo Braun[†], *Justin Luitjens*[†], *Ryan Leary*[†], *Tim Kaldewey*[†], *Daniel Povey*

[†]NVIDIA, Santa Clara, USA

ABSTRACT

We present an optimized weighted finite-state transducer (WFST) decoder capable of online streaming and offline batch processing of audio using Graphics Processing Units (GPUs). The decoder is efficient in memory utilization, input/output (I/O) bandwidth, and uses a novel Viterbi implementation designed to maximize parallelism. The reduced memory footprint allows the decoder to process significantly larger graphs than previously possible, while optimizing I/O increases the number of simultaneous streams supported. GPU preprocessing of lattice segments enables intermediate lattice results to be returned to the requestor during streaming inference. Collectively, the proposed algorithm yields up to a 240x speedup over single core CPU decoding, and up to 40x faster decoding than the current state-of-the-art GPU decoder, while returning equivalent results. This decoder design enables deployment of production-grade ASR models on a large spectrum of systems, ranging from large data center servers to low-power edge devices.

Index Terms— Automatic speech recognition, decoder, WFST, parallel computing, edge

1. INTRODUCTION

Recent advancements in automatic speech recognition (ASR), fueled by deep learning research in the field [1], have led to significant quality improvements, making the technology practical for a slew of human-computer interaction use cases and driving demand for streaming ASR as a service. Streaming ASR as a service typically requires large numbers of commodity servers in a datacenter. Tight latency requirements guided work to improve inference performance of models deployed in datacenters and encouraged research on supporting inference at the edge, including low-power devices [2, 3].

Typical ASR systems comprise three primary components: feature extraction, acoustic modeling, and language model decoding. Historically, the computational complexity of the acoustic model has dominated the inference execution time, and has been the focus of a variety of optimizations, including unusual network architectures, striding, and quantization techniques [4–6].

Principal among these optimizations is offloading acoustic model inference to dedicated acceleration hardware, most commonly GPUs [7]. In many cases, feature extraction and neural acoustic models are efficient enough such that further optimization is limited by Amdahl's law [8]: marginal latency improvements in previously optimized components yield negligible improvements in system latency. To begin our investigation into accelerating speech recognition inference, we profiled a typical lattice decode using the Kaldi speech recognition framework [9] with a pretrained model (see experiments in Section 4), and found 94% of the wallclock time was spent in the language model decoder when using a GPU for acoustic model inference.

In this work, we propose a novel implementation of weighted finite-state transducer (WFST) decoding for the speech recognition task using GPUs and NVIDIA's CUDA [10] programming language. The decoder is designed as a drop-in replacement for existing decoders, requiring no language or acoustic model modifications. It is designed to be maximally flexible, supporting online recognition of multiple simultaneous audio streams and lattice generation. Carefully bounded memory utilization ensures adequate space on GPU memory for large language models and coresident acoustic models. Finally, the algorithm can scale from small GPUs running on low-power embedded GPUs to multiple datacenter-class GPUs running in a single server. Prior to publication, the work has been open-sourced and is now included with Kaldi[1].

2. RELATED WORK

Originally proposed by Mohri [11], WFSTs for ASR decoding have become the de facto standard when using n-gram language models. The decode process returns the single-best path, or alternatively an exact lattice [12] representing multiple possible hypotheses for the decoded utterance. Efforts to increase the speed of the decode and lattice generation process have included parallel, multi-threaded CPU implementations [13] as well as hybrid on-the-fly rescoring [14].

Despite promising efforts in [13], attempts to extend previous accelerated speech decoding onto parallel processors

[1]https://github.com/kaldi-asr/kaldi/tree/master/src/cudadecoder

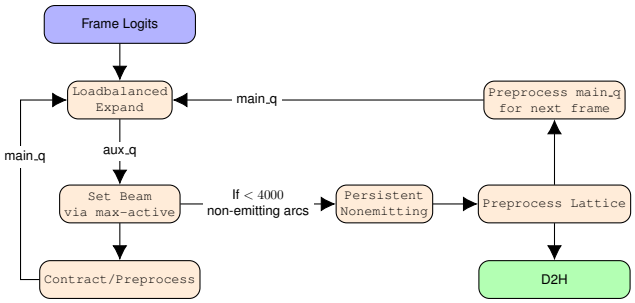

Fig. 1. Block diagram of kernels involved in advancing decoding.

are relatively nascent. Initial efforts targeted hybrid rescoring methods [15, 16] using constrained vocabularies or language models on GPU, while offloading rescoring to CPU. General-purpose WFST decoding on GPUs has been proposed in [17–19], but these works do not support conditioning on acoustic model (AM) posteriors.

The proposed work is most closely related to and improves upon the first fully GPU-accelerated lattice decoder [20], which maps token passing constructs [13] to GPU. Starting from the single-threaded CPU decoder, we tailored the algorithm to the strengths of the hardware, including avoiding unnecessary synchronization and atomics, and using flat, compact memory structures. Efficiencies realized in this implementation enabled the addition of support for online decoding while achieving up to 40x speedups over previous accelerated implementations.

3. PARALLEL VITERBI DECODING

The parallel WFST decoder generally follows the typical order of operations in a serial decoder: for each frame of AM posteriors, the decoder processes emitting arcs (those arcs with non-null labels) conditioned on frame values, processes any chains of non-emitting arcs, and finally performs pruning. The proposed algorithm utilizes two disparate asynchronous CUDA streams: one responsible for executing compute kernels, and the other responsible for performing non-blocking device to host (D2H) memory copies of lattice tokens. Using a second stream for D2H copies makes it possible to return intermediate results during online coding without stalling the compute pipeline.

We eliminate many common CPU-oriented optimizations and constraints, which are sometimes detrimental to parallel performance. Specifically, when expanding tokens, we do not test that new tokens are unique. It is sufficient for correctness to allow duplicate tokens to persist and be cleaned later: trading marginal extra work for reduced dependence on synchronization and atomic operations. Despite further micro-optimizations in the code, we focus this section on the unique architectural decisions of the decoder for brevity.

3.1. Batching & Context Switching

As decoding is necessarily serial in nature (i.e. prediction at time t depends on the state at $t-1$), and individual steps represent relatively small units of work, decoding functions (kernels) executing on the GPU complete quickly, and performance becomes constrained by kernel launch latency. By structuring the decoder such that multiple audio streams are processed in parallel, launch latency is hidden by longer-running kernels (due to their increased workload).

To support efficient decoding for online recognition, we introduce two separate mechanisms for handling simultaneous audio streams: channels and lanes. Lanes are roughly equivalent to batch size in neural networks, and represent the set of utterances or streams being actively decoded. Channels maintain state for utterances which are not ready to continue processing due to lack of audio or computed posteriors. The threaded decoder that readies work for the GPU is responsible for multiplexing channels (as they become ready) onto lanes (as they become available). This scheme allows for easy tuning to match the GPU with the model and representative data: increase the number of lanes until diminishing returns are reached, and set the number of channels to match the measured throughput/xRTF.

Critical to this strategy is the ability to efficiently swap channels with lanes, which requires minimizing memory usage required for state tracking and optimizing layout. In practice, context switching calls complete in about 5µs per batch. Details of the memory structure used is described in the following section.

3.2. Memory Layout

Maximum efficiency depends on minimizing memory usage for state. Equally important is the layout of memory. Careful consideration is taken here to ensure that data is structured such that kernels may use coalesced accesses wherever possible.

3.2.1. Footprint

We represent the decoding FST in-memory as a set of compressed sparse rows (CSRs) and additional metadata, which we are able to efficiently traverse with direct indexing.

Given the decode WFST $T = (\Sigma, \Omega, Q, E, ...)$, with input and output labels Σ and Ω, respectively, a finite set of states Q, a finite set of transitions E (E_E are emitting transitions), we calculate its expected memory utilization, M_{fst} as

$$M_{fst} = 12|Q| + 8|E| + 4|E_E| \quad (1)$$

In practice, this typically equates to GPU memory used for the FST about $\frac{1}{3}$ of the size of the FST on disk.

GPU memory utilization of the decoder is bounded and can be calculated with a closed-form equation based on configured hyperparameters. The memory footprint, in bytes, of

the full state of the decoder, including utterances being actively decoded and those awaiting further decoding is given in Equation 2 where α is the maximum active tokens after pruning (*max-active*), n_l is the maximum number of lanes, and n_c is the maximum number of channels configured.

$$M_{state} = 64\alpha n_c + 544\alpha n_l + 1024 n_l \quad (2)$$

Note that the size of the decoder state is *not* related to the size of the decode graph nor the beam sizes. As such, one can scale the decoder based on the desired number of parallel streams or sizes of the acoustic/language model. As a concrete example, one could configure an edge device for a single stream ($\alpha = 10000, n_c = 1, n_l = 1$) and use only 5.8MB of device memory, while a datacenter-class GPU might support 5000 simultaneous streams in realtime ($\alpha = 10000, n_c = 5000, n_l = 500$) requiring about 5.5GB.

3.3. Load Balancing

To maximize parallelism, it is important that we generate large numbers of threads which have approximately the same amount of work to do. As we process each batch of frames, we begin by performing a load-balanced expand (see Figure 1) where each outgoing arc is processed by its own thread, generating a number of candidate tokens. The adaptive beam is then adjusted, and used to determine which candidates are added back to the main queue for further processing.

Another irregularity comes from the slow convergence of non-emitting iterations, leading to an undefined number of small iterations (i.e. long tail). Once the count of active non-emitting tokens becomes low enough, the following iterations will be processed by a persistent kernel until convergence. In that persistent kernel, each utterance owns only one CUDA Cooperative Thread Array (CTA), speeding up synchronization and intra-thread communication.

3.4. Lattice Preprocessing

Up until the lattice processing stage in the decoder, the goal is to discover which subset of the search space would be saved for the current frame. Following frames build on that subset, and any paths within that subset may be present in the final lattice. During the discovery stage, we had to create and consider (typically an order of magnitude) more tokens than the ones we ultimately keep. Subsequently, the discovery stage focuses on being lightweight, while postponing any expensive structuring operations.

In order to generate a lattice based on these tokens, we convert the raw tokens into a structured CSR representation. This includes detecting tokens linked to the same FST state, listing them in the CSR format, designing a unique representative for each FST state, and computing extra costs. This data is then moved to the host and used to generate the final lattice at the end of utterance. Tokens are then prepared for the next frame by "soft-pruning" any tokens which aren't representative for their FST state by artificially zeroing their out-arc degree, which can then be safely ignored by the load balancer: avoiding exponential growth.

4. EXPERIMENTS

We focus our examination on the performance of two models representing a wide spectrum of deployment conditions: from LibriSpeech [21] *test-clean* subset evaluated with a model tuned specifically for LibriSpeech[2], to the LibriSpeech *test-other* subset evaluated on the ASPiRE [22] Kaldi model[3]. The former represents an ideal case of relatively easy-to-transcribe data being processed by a well-tuned model, while the latter is a more pathological case representing more challenging input audio transcribed by a mismatched model. The net effect of the matched versus mismatched conditions is that in the case of the former, acoustic model posteriors tend to be more confident and fewer paths need to be evaluated when compared to more challenging scenarios. All experiments are performed using a single NVIDIA Tesla V100 GPU, *beam=15*, *lattice-beam=8*, and *max-active=10000*, unless otherwise specified.

4.1. Accuracy

The parallel implementation leads to expected non-determinism, typically due to out-of-order pruning of tokens. Specifically, the histogram pruning thresholds are somewhat arbitrary compared to the explicit cutoff in the baseline implementation. Because of this, we see minor variations in the word error rate ($\pm 0.02\%$).

Decoder	test-clean			test-other		
	lat. den.	WER	OWER	lat. den.	WER	OWER
Baseline	4.19	5.49	1.05	13.94	13.71	2.55
GPU	4.22	5.51	1.09	14.18	13.72	2.67

Table 1. Validation of lattice quality with LibriSpeech model and test sets.

Table 1 evaluates the output lattices against lattices generated by the baseline CPU implementation. We validate that the word error rate (WER) is within tolerable limits, as well as the oracle word error rate (OWER). The Oracle WER is a proxy for determining if all expected alternate paths exist within the lattice. Finally, we measure the lattice density (*lat. den.*), which is an average measure of outgoing arcs. This confirms the produced lattices are of similar size.

4.2. Speed Improvements

Table 2 reports xRTF (times faster than real time) for baseline Kaldi single- and multi-process decoder implementations, and other GPU decoder implementations. The CPU

[2]Using standard Kaldi LibriSpeech recipe

[3]Available from http://kaldi-asr.org/models/m1

speeds are obtained using an Intel Xeon CPU E5-2698 v4 @ 2.20GHz, with 20 cores.

Across the tested configurations, the GPU decoder outperforms the multithreaded CPU implementation within Kaldi, with a relative speedup ranging between 14x and 18x when compared to a full 20-core Xeon processor. When compared with the current state-of-the art parallel decoder [20], the proposed algorithm decodes between 11x and 41x faster.

Decoder	Type	ASPiRE		LibriSpeech	
		clean	other	clean	other
CPU Process	One Best	4.4	2.9	57.2	26.0
CPU Process	Lattice	3.8	2.7	53.4	29.2
CPU Socket	Lattice	43.2	30.1	614.8	313.1
GPU [20]	Lattice	70.9	n/a	219.9	174.6
GPU *(This Work)*	Lattice	**769.3**	**649.7**	**9 031.4**	**4 391.7**

Table 2. Offline decoding speed (xRTF, *beam=15*)[4].

4.3. Hyperparameters

Decoding hyperparameter selection (particularly *beam*) impacts decoder speed. In cases with smaller beam widths, oversubscription of threads to the GPU is reduced, enabling faster inference. Care should be taken to choose a beam width that is suitable for the target data and model. Figure 2 shows a roughly log-linear decrease in decode speed as beam width increases. The points in the graph are labeled with WER at that operating point. Note the marginal accuracy improvements despite significant increases in runtime.

LM	HCLG	test-clean		test-other	
	Size (MB)	xRTF	WER	xRTF	WER
3-gram, 3e-10	192.6	5.51	9 031.4	13.72	4 391.7
3-gram, 1e-10	467.0	4.92	9 064.5	12.54	4 386.8
3-gram	8724.0	**4.02**	**9 161.7**	**10.09**	**4 627.4**

Table 3. Comparison of FST size and WER/Speed.

Table 3 shows that significant reductions in WER may be achieved by using larger language models. Three different trigram language models with different pruning thresholds (3e−10, 1e−10, and no pruning, respectively) are used with other parameters held constant. Despite a 10x filesize difference, the decode performs *faster* using the large language model likely due to reduced perplexity during decoding yielding extra pruning, and subsequently improved speed.

4.4. Deployment

With fully GPU-accelerated inference, the CPU is only left responsible for shuffling data in/out of the GPU, and completing lattice determinization if required. Because of this, multi-GPU scaling is nearly linear. On a NVIDIA DGX-1 containing 8 V100 GPUs, 85% scaling efficiency is achieved when using all GPUs.

[4]Missing data for prior GPU implementation is due to application crashes.

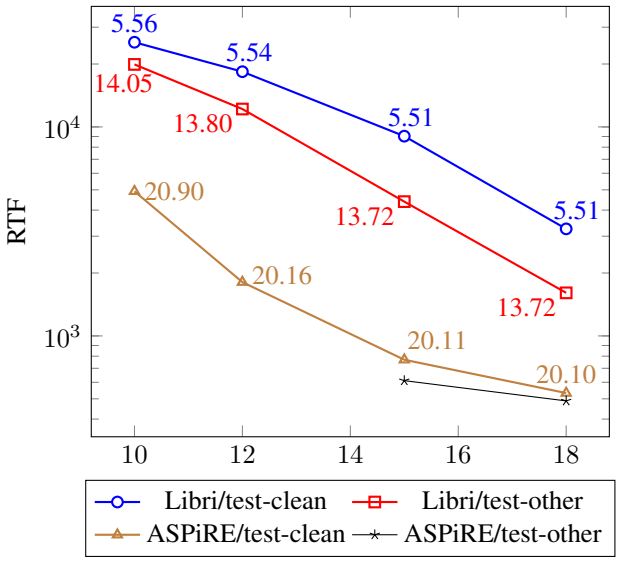

Fig. 2. RTF vs beam width.

GPU	Class	Streams (10)	Streams (15)	TDP
Jetson Nano	Embedded	11	7	5
AGX Xavier	Embedded	502	399	30
Tesla T4	Datacenter	2024	1561	70
Tesla V100	Datacenter	4117	3150	250

Table 4. Measured end-to-end realtime throughput across suite of NVIDIA GPUs at varying beam sizes.

Table 4 demonstrates the same decoder used across the entire current NVIDIA family of processors. In all cases, the models are identical, and use the same hyperparameters except for batch size. The values in the table represent the number of streams that can be decoded in realtime, and includes feature extraction and acoustic model.

5. CONCLUSION

In this paper, we present a parallel decoder for speech recognition WFST inference. The algorithm is AM and LM agnostic, requiring no changes to support inference with existing models trained in the Kaldi toolkit. By implementing the decoder such that multiple utterances are processed in parallel, optimized memory management, and trading extra computation for reduced synchronization, we consistently achieve order-of-magnitude speedups when compared to the baseline multithreaded algorithm on CPU and current state-of-the-art GPU implementation. We further demonstrate that this work can be used on embedded platforms without requiring any model changes.

The implementation is now open-source as part of the Kaldi release. Future work will evaluate adaptations for CTC decoding as well as adding support for on-the-fly neural language model scoring.

6. BIBLIOGRAPHY

[1] G. Hinton, L. Deng, D. Yu, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[2] Y. He, T. N. Sainath, R. Prabhavalkar, *et al.*, "Streaming end-to-end speech recognition for mobile devices," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2019, pp. 6381–6385.

[3] I. McGraw, R. Prabhavalkar, R. Alvarez, *et al.*, "Personalized speech recognition on mobile devices," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2016, pp. 5955–5959.

[4] G. Pundak and T. Sainath, "Lower frame rate neural network acoustic models," in *Interspeech*, 2016.

[5] X. Xiang, Y. Qian, and K. Yu, "Binary deep neural networks for speech recognition," Aug. 2017, pp. 533–537.

[6] V. Peddinti, Y. Wang, D. Povey, and S. Khudanpur, "Low latency acoustic modeling using temporal convolution and lstms," *IEEE Signal Processing Letters*, vol. 25, no. 3, pp. 373–377, Mar. 2018.

[7] P. R. 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, Oct. 2009.

[8] D. P. Rodgers, "Improvements in multiprocessor system design," *SIGARCH Comput. Archit. News*, vol. 13, no. 3, pp. 225–231, Jun. 1985.

[9] D. Povey, A. Ghoshal, G. Boulianne, *et al.*, "The kaldi speech recognition toolkit," in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, IEEE Catalog No.: CFP11SRW-USB, Hilton Waikoloa Village, Big Island, Hawaii, US: IEEE Signal Processing Society, Dec. 2011.

[10] NVIDIA, *Cuda toolkit documentation*.

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

[12] D. Povey, M. Hannemann, G. Boulianne, *et al.*, "Generating exact lattices in the wfst framework," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2012, pp. 4213–4216.

[13] C. Mendis, J. Droppo, S. Maleki, *et al.*, "Parallelizing wfst speech decoders," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2016, pp. 5325–5329.

[14] T. Hori, C. Hori, and Y. Minami, "Fast on-the-fly composition for weighted finite-state transducers in 1.8 million-word vocabulary continuous speech recognition," in *INTERSPEECH*, 2004.

[15] H. Sak, M. Saraçlar, and T. Güngör, "On-the-fly lattice rescoring for real-time automatic speech recognition," in *INTERSPEECH*, 2010.

[16] J. Kim, J. Chong, and I. R. Lane, "Efficient on-the-fly hypothesis rescoring in a hybrid gpu/cpu-based large vocabulary continuous speech recognition engine," in *INTERSPEECH*, 2012.

[17] D. Zhang, R. Zhao, L. Han, *et al.*, "An implementation of viterbi algorithm on gpu," in *Proceedings of the 2009 First IEEE International Conference on Information Science and Engineering*, ser. ICISE '09, Washington, DC, USA: IEEE Computer Society, 2009, pp. 121–124.

[18] M. K. Hanif and K.-H. Zimmermann, "Accelerating viterbi algorithm on graphics processing units," *Computing*, vol. 99, pp. 1105–1123, 2017.

[19] A. Argueta and D. Chiang, "Decoding with finite-state transducers on GPUs," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, Valencia, Spain: Association for Computational Linguistics, Apr. 2017, pp. 1044–1052.

[20] Z. Chen, J. Luitjens, H. Xu, *et al.*, "A gpu-based wfst decoder with exact lattice generation," in *INTERSPEECH*, 2018.

[21] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An asr corpus based on public domain audio books," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Apr. 2015, pp. 5206–5210.

[22] M. Harper, "The automatic speech recogition in reverberant environments (aspire) challenge," in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, Dec. 2015, pp. 547–554.