A Fully Data Parallel WFST-based Large Vocabulary Continuous Speech Recognition on a Graphics Processing Unit

Jike Chong, Ekaterina Gonina, Youngmin Yi, Kurt Keutzer

Department of Electrical Engineering and Computer Science, University of California, Berkeley
`{jike, egonina, ymyi, keutzer}@eecs.berkeley.edu`

Abstract

Tremendous compute throughput is becoming available in personal desktop and laptop systems through the use of graphics processing units (GPUs). However, exploiting this resource requires re-architecting an application to fit a data parallel programming model. The complex graph traversal routines in the inference process for large vocabulary continuous speech recognition (LVCSR) have been considered by many as unsuitable for extensive parallelization. We explore and demonstrate a fully data parallel implementation of a speech inference engine on NVIDIA's GTX280 GPU. Our implementation consists of two phases - compute-intensive observation probability computation phase and communication-intensive graph traversal phase. We take advantage of dynamic elimination of redundant computation in the compute-intensive phase while maintaining close-to-peak execution efficiency. We also demonstrate the importance of exploring application-level trade-offs in the communication-intensive graph traversal phase to adapt the algorithm to data parallel execution on GPUs. On 3.1 hours of speech data set, we achieve more than 11× speedup compared to a highly optimized sequential implementation on Intel Core i7 without sacrificing accuracy.

Index Terms: Data parallel, Continuous Speech Recognition, Graphics Processing Unit

1. Introduction

Graphics processing units (GPUs) are enabling tremendous compute capabilities in personal desktop and laptop systems. Recent advances in the programming model for GPUs such as CUDA [1] from NVIDIA have provided an implementation path for many exciting applications beyond graphics processing such as speech recognition. In order to take advantage of high throughput capabilities of the GPU-based platforms, programmers need to transform their algorithms to fit the data parallel model. This can be challenging for algorithms that don't directly map onto the model, such as graph traversal in a speech inference engine.

In this paper we explore the use of GPUs for large vocabulary continuous speech recognition (LVCSR) on NVIDIA GTX280 GPU. A LVCSR application analyzes a human utterance from a sequence of input audio waveforms to interpret and distinguish the words and sentences intended by the speaker. Its top level architecture is shown in Fig. 1. The recognition process uses a Weighted Finite State Transducer (WFST) based *recognition network* [2], which is a language database that is compiled offline from a variety of knowledge sources using powerful statistical learning techniques. The *speech feature extractor* collects feature vectors from input audio waveforms, and then the Hidden-Markov-Model-based *inference engine* computes the most likely word sequence based on the extracted speech features and the recognition network. In the LVCSR system the common speech feature extractors can be parallelized using standard signal processing techniques. On the other hand, the graph traversal routines have been considered unsuitable for extensive parallelization [3, 4]. This paper discusses the application-level trade-offs that need to be made in order to efficiently parallelize the graph traversal process in LVCSR and illustrates the performance gains obtained from effective parallelization of this portion of the algorithm.

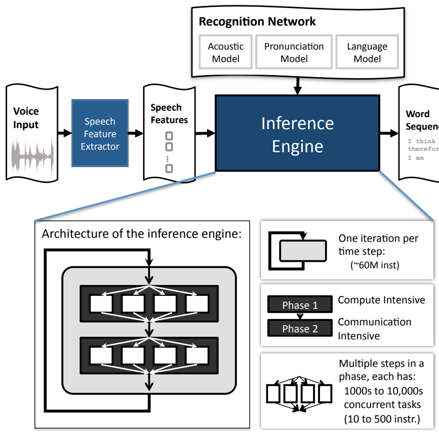

Figure 1: Architecture of large vocabulary continuous speech recognition

A parallel inference engine traverses a graph-based knowledge network consisting of millions of states and arcs. As shown in Fig. 1, it uses the Viterbi search algorithm to iterate through a sequence of input audio features one time step at a time [5]. The Viterbi search algorithm keeps track of each alternative interpretation of the input utterance as a sequence of states ending in an active state at the current time step. It evaluates out-going arcs based on the current-time-step observation to arrive at the next set of active states. Each time step consists of two phases: Phase 1 - observation probability computation and Phase 2 - graph traversal computation. Phase 1 is compute-intensive while Phase 2 is communication-intensive.

The inference engine is implemented using CUDA, which requires the computation to be organized into a sequential host program on a CPU calling parallel kernels running on the GPU. A kernel executes a scalar sequential program across a set of parallel threads where each thread operates on a different piece of data. The CPU and the GPU have separate memory spaces and there is an implicit global barrier between different kernels, as illustrated at the bottom of Fig. 1.

Managing aggressive pruning techniques to keep LVCSR computationally tractable requires frequent global synchronizations. We are keeping track of on average 0.1-1.0% of the total state space and must communicate the pruning bounds every time step. There exist significant parallelism opportunities in each time step of the inference engine. For example, we can evaluate thousands of alternative interpretations of a speech utterance concurrently. At the same time, the inference engine involves a parallel graph traversal through a highly irregular knowledge network. The traversal is guided by a sequence of input audio features that continuously changes the working set at run time. The challenge is to not only define a software architecture that exposes sufficient fine-grained application concurrency (Section 3.1), but also to extract close-to-peak performance on the GPU platform (Section 3.2). We also explore alternatives in the recognition network structure for more efficient execution on a data parallel implementation platform (Section 3.3).

2. Related Work

There have been many efforts in paralleling LVCSR. We highlight three categories of efforts in software-based acceleration.

Category 1: Data Parallel, multiprocessor shared memory implementation on multiprocessor clusters [6, 7]. These implementations are plagued by high communication overhead in the platform, high sequential overhead in the software architecture, load imbalance among parallel tasks or excessive memory bandwidth and thus are limited in scalability to more parallel platforms. In [8] some of these issues were resolved by using OpenMP as an implementation platform, however, it was based on the tree-lexicon search network, a less efficient approach than the WFST-based approach [2] used in this paper.

Category 2: Task parallel implementation. Ishikawa *et al.* [9] exploited pipelined task-level parallelism on three ARM cores. Here, scaling required extensive redesign effort.

Category 3: Data Parallel implementation on manycore accelerator in CPU-based host systems [10, 11, 12]. [10, 11] focused on speeding up the compute intensive phase but left the communication intensive phases on the host platform, thereby limiting their scalability. [12] leveraged the simpler structure of a linear-lexicon based (LL) recognition network to achieve a 9× speedup compared to a highly optimized sequential implementation. However, LL-based recognition networks are less efficient than the WFST-based recognition networks [13, 14].

In contrast, we optimized our software architecture by implementing data parallel versions of both of the observation probability computation and the graph traversal phases for a GPU platform. We used the more challenging WFST-based recognition network and achieved greater speedups in each of the two phases.

3. Data Parallel Inference Engine

Among the two phases of the inference engine shown in Fig. 1, the compute-intensive phase involves using Gaussian Mixture Models (GMM) to estimate the likelihood that an input audio feature matches a triphone state. This phase maps well to highly parallel platforms such as GPUs. The communication-intensive phase involves traversing through a highly irregular recognition network, while managing a dynamic working set that changes every time step based on input audio features. Although this phase is highly challenging to implement on parallel platforms, we demonstrate that with carefully managed application-level trade-offs, significant speedups can still be achieved.

3.1. Overall Optimizations

Implementing both phases on the GPU has significant advantages over separatly implementing the compute-intensive phase on the GPU and the communication-intensive phase on the CPU. A split implementation incurs high data-copying overheads between the CPU and the GPU for transfering large amounts of intermediate results. It is also less scalable as the transfers become a sequential bottleneck in the algorithm. Implementing all phases to run on the GPU eliminates data transfers between the CPU and the GPU and allows for more scalable parallel pruning routines.

Both of the phases extensively use the vector units on the GPU, which require *coalesced* memory accesses and *synchronized* instruction execution. Memory accesses are *coalesced* when data is referenced from consecutive memory locations so it can be loaded and used in a vector arithmetic unit directly without rearrangement. The kernels are written to have *synchronized* instruction control flow so that all lanes in a vector unit are doing useful work while executing the same operation, i.e. the Single-Instruction-Multiple-Data (SIMD) approach.

To maximize *coalesced* memory accesses, we create a set of buffers to gather the active state and arc information from the recognition network at the beginning of each time step for all later references in that time step. In addition, we use arc-based traversal where each SIMD lane is assigned to compute one out-going arc. Since the amount of computation is the same for all out-going arcs, all SIMD lanes are *synchronized* during this computation. This approach yields more efficient SIMD utilization and results in 5× performance gain for the communication intensive phase compared to traversing the graph with one state per SIMD lane, where each lane has different amount of work depending on the number of out-going arcs a state has.

To coordinate the graph traversal across cores, we extensively use atomic operations on the GPU. When computing the arc with the most-likely incoming transition to a destination state, each arc transition updates a destination state atomically. This efficiently resolves write-conflicts when multiple cores compute arcs that share the same destination state.

3.2. Compute-intensive Phase Optimization

In the compute-intensive phase, we compute the observation probability of triphone states. This involves two steps: (1) GMM computation and (2) logarithmic mixture reduction. Our implementation distributes the clusters across GPU cores and uses parallel resources within-core to compute each cluster's mixture model. Both steps scale well on highly parallel processors and the optimization is in eliminating redundant work.

A typical recognition network has millions of arcs, each labeled with one of the approximately 50,000 triphone states. Furthermore, the GMM for the triphone states can be clustered into 2000-3000 clusters. In each time step, on average only 60% of the clusters and 20% of the triphone states are used.

We prune the list of GMM and triphone states to be computed in each time step based on the lexicon model compiled into the WFST recognition network. We remove the redundant GMM and triphone states from consideration for each time step, thereby reducing the computation time for this phase by 70%.

3.3. Communication-intensive Phase Optimizations

The communication-intensive phase involves a graph traversal process through an irregular recognition network. There are two types of arcs in a WFST-based recognition network: arcs with an input label (non-epsilon arcs), and arcs without input labels (epsilon arcs). In order to compute the set of next states in a given time step, we must traverse both the non-epsilon and all the levels of epsilon arcs from the current set of active states. This multi-level traversal can impair performance significantly. We explore the modification of flattening the recognition network to reduce the number of levels of arcs that need to be traversed and observe corresponding performance improvements. To illustrate this, Fig. 2 shows a small section of a WFST-based

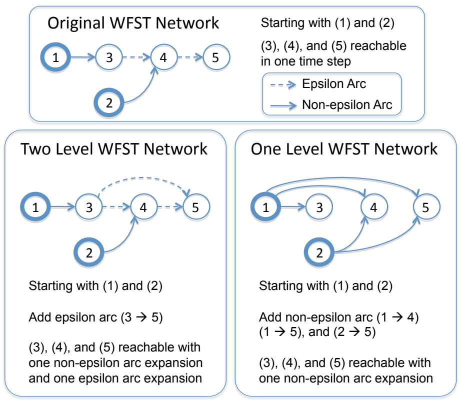
Original WFST Network
1
3
4
5
2
Starting with (1) and (2)
(3), (4), and (5) reachable in one time step
Epsilon Arc
Non-epsilon Arc
Two Level WFST Network
Starting with (1) and (2)
Add epsilon arc (3 → 5)
(3), (4), and (5) reachable with one non-epsilon arc expansion and one epsilon arc expansion
One Level WFST Network
Starting with (1) and (2)
Add non-epsilon arc (1 → 4) (1 → 5), and (2 → 5)
(3), (4), and (5) reachable with one non-epsilon arc expansion

Figure 2: Network modification techniques for a data parallel inference engine

recognition network. Each time step starts with a set of currently active states, e.g. states (1) and (2) in Fig. 2, representing the alternative interpretations of the input utterances. It proceeds to evaluate all out-going non-epsilon arcs to reach a set of destination states, e.g. states (3) and (4). The traversal then extends through epsilon arcs to reach more states, e.g. state (5) for the next time step.

The traversal from state (1) and (2) to (3), (4) and (5) can be seen as a process of active state wave-front expansion in a time step. The challenge for data parallel operations is that the expansion from (1) to (3) to (4) to (5) requires multiple level traversal: one non-epsilon level and two epsilon levels. The traversal incurs significant instruction stream divergence and uncoalesced memory accesses if each thread expands through an arbitrary number of levels. Instead, a data parallel traversal limits the expansion to one level at a time and the recognition network is augmented such that the expansion process can be achieved with a fixed number of single level expansions. Fig.2 illustrates the necessary recognition network augmentations for Two-Level and One-Level setups.

Each step of expansion incurs some overhead, so to reduce the fixed cost of expansion we want fewer number of steps in the traversal. However, depending on recognition network topology, augmenting the recognition network may cause significant increase in the number of arcs in the recognition network, thus increasing the variable cost of the traversal. We demonstrate this trade-off with a case study in the Results section.

4. Results

4.1. Experimental Platform and Baseline Setup

We use the NVIDIA GTX280 GPU with a Intel Core2 Q9550 based host platform. GTX280 has 30 cores with 8-way vector arithmetic units running at 1.296GHz. The processor architecture allows a theoretical maximum of 3 floating point operations (FLOP) per cycle, resulting in a maximum of 933 GFLOP of peak performance per second. The sequential results were measured on an Intel Core i7 920 based platform with 6GB of DDR memory. The Core i7-based system is 30% faster than the Core2-based system because of its improved memory sub-system, providing a more conservative speedup comparison. The sequential implementation was compiled with icc 10.1.015 using all automatic vectorization options. Kernels in the compute-intensive phase were hand optimized with SSE intrinsics [15]. As shown in Fig. 3, the sequential performance achieved was 3.23 seconds per one second of speech, with Phase 1 and 2 taking 2.70 and 0.53 seconds respectively. The parallel implementation was compiled with icc 10.1.015 and nvcc 2.2 using Compute Capability v1.3.

4.2. Speech Models and Test Sets

The speech models were taken from the SRI CALO real-time meeting recognition system [16]. The frontend uses 13d PLP features with 1st, 2nd, and 3rd order differences, VTL-normalized and projected to 39d using HLDA. The acoustic model was trained on conversational telephone and meeting speech corpora, using the discriminative MPE criterion. The LM was trained on meeting transcripts, conversational telephone speech and web and broadcast data [17]. The acoustic model includes 52K triphone states which are clustered into 2,613 mixtures of 128 Gaussian components. The recognition network is an $H \circ C \circ L \circ G$ model compiled using WFST techniques [15].

The test set consisted of excerpts from NIST conference meetings taken from the "individual head-mounted microphone" condition of the 2007 NIST Rich Transcription evaluation. The segmented audio files total 3.1 hours in length and comprise 35 speakers. The meeting recognition task is very challenging due to the spontaneous nature of the speech[1]. The ambiguities in the sentences require larger number of active states to keep track of alternative interpretations which leads to slower recognition speed.

Table 2: Accuracy, word error rate (WER), for various beam sizes and decoding speed in real-time factor (RTF)

Avg. # of Active States		32398	19306	9763	3390
WER		51.1	50.9	51.4	54.0
RTF	Sequential CPU	4.36	3.17	2.29	1.20
	Parallel GPU	0.37	0.29	0.20	0.14
Speedup		11.7	11.0	11.3	9.0

Our recognizer uses an adaptive heuristic to control the number of active states by adjusting the pruning threshold at run time. This allows all traversal data to fit within a pre-allocated memory space. Table 2 shows the decoding accuracy, i.e., word error rate (WER) with varying thresholds and the corresponding decoding speed on various platforms. The recognition speed is represented by the real-time factor (RTF) which is computed as the total decoding time divided by input speech duration.

As shown in Table 2, the GPU implementation can achieve order of magnitude more speedup over the sequential implementation [15] for the same number of active states. More importantly, one can trade-off speedup with accuracy. For example, one can achieve 54.0% WER traversing an average of 3390 states per time step with a sequential implementation, or one can achieve a 50.9% WER traversing an average of 19306 states per time step while still getting a 4.1× speedup, improving from an RTF of 1.20 to 0.29.

4.3. Compute-intensive Phase

The parallel implementation of the compute-intensive phase achieves close to peak performance on GTX280. As shown in Table 3, we found the GMM computation memory-bandwidth-limited and our implementation achieves 85% of peak memory bandwidth. The logarithmic mixture reduction is compute-limited and our implementation achieves 98% of achievable peak compute performance given the instruction mix.

4.4. Communication-intensive Phase

Parallelizing this phase on a quadcore CPU achieves a 2.85× performance gain [15] and incurs intermediate result transfer overhead. We achieved a 3.84× performance gain with an equivalent configuration on the GPU and avoided intermediate

[1]A single-pass time-synchronous Viterbi decoder from SRI using lexical tree search achieves 37.9% WER on this test set

Table 1: Performance with Different Recognition Network Augmentation (Run times Normalized to one Second of Speech)

Active States	Original WFST Network				Two-Level WFST Network				One-Level WFST Network			
(% of state space)	0.1%	0.3%	0.5%	1.0%	0.1%	0.3%	0.5%	1.0%	0.1%	0.3%	0.5%	1.0%
Total States	4,114,507				4,114,672 (+0.003%)				4,116,732 (+0.05%)			
Total Arcs	9,585,250				9,778,790 (+2.0%)				12,670,194 (+32.2%)			
Arcs Traversed*	27,119	64,489	112,173	171,068	27,342	65,218	114,043	173,910	44,033	103,215	174,845	253,339
Arcs increase (%)	-	-	-	-	+0.8%	+1.1%	+1.7%	+1.7%	+62%	+60%	+56%	+48%
Phase 1 (ms:%)	77:41%	112:43%	146:43%	177:41%	77:48%	112:50%	146:48%	178:45%	73:55%	110:55%	147:51%	177:48%
Phase 2 (ms:%)	97:52%	127:49%	171:50%	230:53%	74:46%	99:44%	138:45%	191:48%	52:39%	81:40%	125:43%	175:47%
Seq. Ovrhd (ms:%)	13:7%	20:8%	24:7%	28:6%	11:7%	14:6%	20:7%	25:6%	8: 6%	10:5%	16:5%	21:6%
Total (ms)	187	258	341	436	161	225	304	393	134	202	289	373
Faster than real time	5.3×	3.9×	2.9×	2.3×	6.2×	4.4×	3.3×	2.5×	7.5×	5.0×	3.5×	2.7×

* *Average number of arcs traversed per time step*

Table 3: Efficiency of the Computation Intensive Phase

(GFLOP/s)	Step 1	Step 2
Theoretical Peak	933	933
	Mem BW Limited	Inst Mix Limited
Practical Peak	227	373
Measured	194	367
Utilization	85%	98%

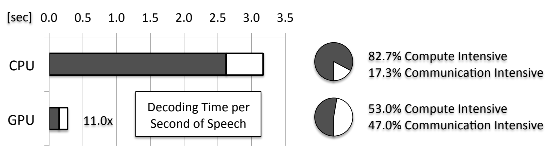

Figure 3: Parallel Speedup of the Inference Engine

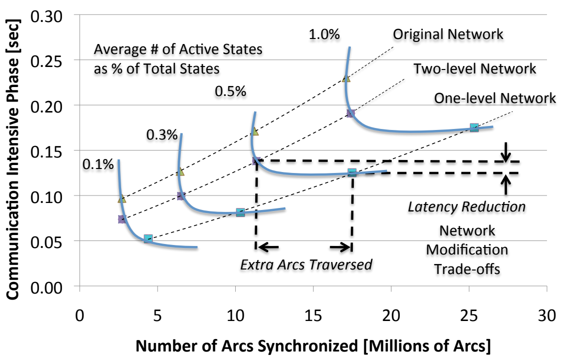

Figure 4: Communication Intensive Phase Run Time in the Inference Engine (normalized to one second of speech)

result transfer overhead. Despite the better speedup on the GPU, this phase became more dominant as shown in Fig 3.

Table 1 demonstrates the trade-offs of recognition network augmentation for efficient data parallel traversal in our inference engine. The augmentation for Two-Level setup resulted in a 2.0% increase in arc count and the augmentation for One-Level setup resulted a 32.2% increase. The dynamic number of arcs evaluated increased marginally for the Two-Level setup. However for the One-Level solution it increased significantly by 48-62%, as states with more arcs were visited more frequently.

Fig 4 shows the run time for various pruning thresholds. The network modifications are described in Section 3.3. Without network modifications, there is significant performance penalty as multiple levels of epsilon arcs must be traversed with expensive global synchronization steps between levels. With minimal modifications to the network, we see a 17-24% speedup for this phase. An additional 8-29% speedup can be achieved by eliminating epsilon arcs completely, saving the fixed cost of one level of global synchronization routines, but this comes at the cost of traversing 48-62% more arcs.

5. Conclusion

We presented a fully data parallel speech inference engine[2] with both observation probability computation and graph traversal implemented on an NVIDIA GTX280 GPU. Our results show that modifications to the recognition network are essential for effective implementation of data parallel WFST-based LVCSR algorithm on GPU. Our implementation achieved up to 11.7× speedup compared to highly optimized sequential implementation with 5-8% sequential overhead without sacrificing accuracy. This software architecture enables performance improvement potentials on future platforms with more parallelism.

[2]Thanks to Kisun You, Nelson Morgan, Adam Janin and Andreas Stolcke for insightful discussions, and to NVIDIA for hardware donation. This research is supported in part by an Intel Ph.D. Research Fellowship, by Microsoft (Award #024263), by Intel (Award #024894), and by matching fund from U.C. Discovery (Award #DIG07-10227).

6. References

[1] *NVIDIA CUDA Programming Guide*, NVIDIA Corporation, 2009, version 2.2 beta. [Online]. Available: http://www.nvidia.com/CUDA

[2] M. Mohri, F. Pereira, and M. Riley, "Weighted finite state transducers in speech recognition," *Computer Speech and Language*, vol. 16, 2002.

[3] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. Berry, "Challenges in parallel graph processing," *Parallel Processing Letters*, 2007.

[4] A. Janin, "Speech recognition on vector architectures," Ph.D. dissertation, University of California, Berkeley, Berkeley, CA, 2004.

[5] H. Ney and S. Ortmanns, "Dynamic programming search for continuous speech recognition," *IEEE Signal Processing Magazine*, vol. 16, 1999.

[6] M. Ravishankar, "Parallel implementation of fast beam search for speaker-independent continuous speech recognition," 1993.

[7] S. Phillips and A. Rogers, "Parallel speech recognition," *Intl. Journal of Parallel Programming*, vol. 27, no. 4, pp. 257–288, 1999.

[8] K. You, Y. Lee, and W. Sung, "OpenMP-based parallel implementation of a continous speech recognizer on a multi-core system," in *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, Taipei, Taiwan, 2009.

[9] S. Ishikawa, K. Yamabana, R. Isotani, and A. Okumura, "Parallel LVCSR algorithm for cellphone-oriented multicore processors," in *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, Toulouse, France, 2006.

[10] P. R. Dixon, T. Oonishi, and S. Furui, "Fast acoustic computations using graphics processors," in *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, Taipei, Taiwan, 2009.

[11] P. Cardinal, P. Dumouchel, G. Boulianne, and M. Comeau, "GPU accelerated acoustic likelihood computations," in *Proc. Interspeech*, 2008.

[12] J. Chong, Y. Yi, N. R. S. A. Faria, and K. Keutzer, "Data-parallel large vocabulary continuous speech recognition on graphics processors," in *Proc. Workshop on Emerging Applications and Manycore Architectures*, 2008.

[13] X. Huang, A. Acero, and H.-W. Hon, *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*. Prentice-Hall, 2001.

[14] S. Kanthak, H. Ney, M. Riley, and M. Mohri, "A comparison of two LVR search optimization techniques," in *Proc. Intl. Conf. on Spoken Language Processing (ICSLP)*, Denver, Colorado, USA, 2002, pp. 1309–1312.

[15] J. Chong, K. You, Y. Yi, E. Gonina, C. Hughes, W. Sung, and K. Keutzer, "Scalable HMM based inference engine in large vocabulary continuous speech recognition," *Workshop on Multimedia Signal Processing and Novel Parallel Computing*, July 2009.

[16] G. Tur *et al.*, "The CALO meeting speech recognition and understanding system," in *Proc. IEEE Spoken Language Technology Workshop*, 2008.

[17] A. Stolcke, X. Anguera, K. Boakye, O. Cetin, A. Janin, M. Magimai-Doss, C. Wooters, and J. Zheng, "The SRI-ICSI Spring 2007 meeting and lecture recognition system," *Lecture Notes in Computer Science*, vol. 4625, no. 2, pp. 450–463, 2008.