**Automatic Speech Recognition (ASR)**

- Automatic speech recognition (ASR) allows multimedia content to be transcribed from audio waves to words.
- This is a challenging task as there can be exponentially many ways to interpret an utterance (a sequence of phones) into words.
- ASR uses the hidden Markov model (HMM).
- States are hidden because phones are instantly observed through the audio.
- Must infer the most likely interpretation while taking the language model into account.

**ASR Characteristics and Software Architecture**

- The Viterbi algorithm is used to infer the most likely interpretation of the observed waveform.
- The forward pass and a backward pass are used to compute the probability.
- The two passes of execution have duplicate sources of the same hidden Markov model.
- Parallelism is composed using Weighted Finite State Transducer (WFST) techniques.
- In a typical recognition sequence, only about 1% to 3% of the hidden Markov model transitions are traversed.

**Challenge 1: Handling irregular data structures with data-parallel operations**

- In the forward pass of the Viterbi algorithm, there are 1,000s to 10,000s of concurrent tasks that represent the most likely interpretations of the hidden Markov model.
- To track these alternative interpretations, one has to reference a selected subset of the hidden Markov model.
- The concurrent access of irregular data structures requires uncoalesced memory accesses to the main memory implementation algorithm steps, which degrades performance.

**Solution 1: Construct efficient dynamic vector data structure to handle irregular data accesses**

- Implement efficient data-parallel operations to handle irregular data structures.
- An ASR application extracts features from a waveform, compares them to the recognition network, and infers the most likely word sequence.
- The recognition network is compiled off-line from a variety of knowledge sources.
- The inference process traverses a graph-based recognition network using the Viterbi algorithm.
- This architecture is modular and flexible.
- It can be adapted to recognize different languages by swapping in different recognition networks and different speech engines.
- Caching them in a memory-aware coalesced runtime data structure allows any parallel application to operate without performance penalties.

**Challenge 2: Eliminating redundant work when threads are computing results for an unpredictable subset of the problems based on input**

- The forward pass in the Viterbi algorithm requires 10,000 – 20,000 alternative steps in the two phases.
- The two phases of execution dominate each iteration of the Viterbi algorithm.
- The software architecture presents significant challenges when implemented on the GPU, see below for details.

**Solution 2: Implement efficient find-unique function by leveraging the GPU global memory write-contrast-resolution policy**

- The traditional approach for finding unique elements in a data-parallel algorithm involves inserting Coping copied unique elements.
- In a temporal network, there are millions of states, each labeled with one of 10,000 hidden Markov model labels.
- A traditional approach with 100,000 possible labels, there can be a hash table of all possible labels.
- A finer-grained concurrency in ASR lies in the evaluation of each alternative step in the forward phase and phase 2.
- The fine-grained concurrency in ASR lies in the evaluation of each alternative step in the forward phase and phase 2.
- The two phases of execution dominate each iteration of the Viterbi algorithm.
- The software architecture presents significant challenges when implemented on the GPU, see below for details.

**Challenge 3: Conflict-free reduction in graph traversal to implement the Viterbi beam-search algorithms**

- During graph traversal, active states are being processed in parallel.
- Different conflicting frequently occurring operations are trying to update the same destination state.
- Each iteration of the Viterbi algorithm involves updating the hidden Markov model.
- The most likely result for each state is essential for achieving good performance.

**Solution 3: Implement lock-free accesses of a shared map leveraging advanced GPU atomic operations to enable conflict-free reductions**

- CUDA atomic operations with various flavors of atomic operations.
- The “atomically” operation is ideal for statistical inference.
- By using work-shedding, the result in each atomically accessed memory location will be the average of all results that was attempted to be written to that location.

**Challenge 4: Parallel construction of a shared queue while avoiding sequential bottlenecks when atomically accessing queue control variables**

- When many threads are trying to insert tasks into a global task queue, significant parallelism can be lost at the point of queue control.
- CUDA guarantees at least one conflicting write to a device memory location to be successful, which is enough to build a flag array.
- The “atomically” operation is ideal for statistical inference.

**Solution 4: Use of hybrid local/global atomic operations and local buffers for the construction of a shared global queue to avoid sequential bottlenecks in accessing global queue control variables**

- By using hybrid global/local queues, we can eliminate the single-point of serialization.
- Each sub-problem can build up its local queue using local atomic operations, which are much faster and can be performed in one batch process, and thus are significantly more efficient.

---

**Results**

- The speech model is taken from the STR-CALaunch evaluation recognition system.
- The acoustic model includes 32,000 triangle subbands classified into 2,615 mixtures of 128 Gaussian components.
- The pronunciation model contains 59,000 words with 88,000 pronunciations.
- The Pronunciation model is using a large vocabulary model with 167k bigram transitions was used.
- Results presented were based on 9500 hours of training.
- An order of magnitude speedup was achieved as compared to an SIMD optimized sequential implementation running on one core of Core 2 processor.
- The communication-intensive phase was accelerated by 3.7x.

---

**Key Lessons**

- **Challenge 1:** Having the freedom to improve the data layout of the runtime data structures is crucial to effectively exploit the fine-grained concurrency in ASR.
- **Challenge 2:** An effective sequential algorithm cannot be directly translated into a parallel algorithm. Memoization does not have an equivalent efficient parallel form. The work-division approach for finding unique elements in a data structure has been dramatically modified to execute efficiently on a GPU.
- **Challenge 3:** Hardware atomic operation support is extremely important for highly parallel applications.
- **Challenge 4:** Local synchronization scope important for leveraging global synchronization bottlenecks.