Efficient Automatic Speech Recognition on the GPU

GPU Technology Conference

September 23, 2010

Jike Chong

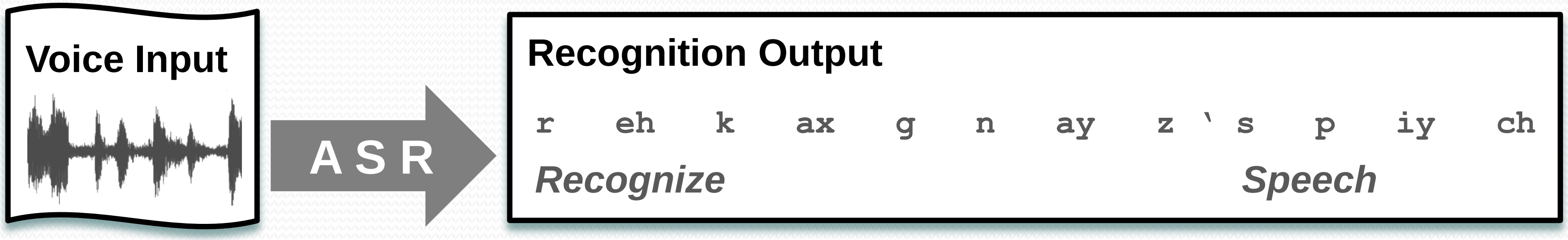
Principal Architect

Parasians, LLC

Speaker: Jike Chong

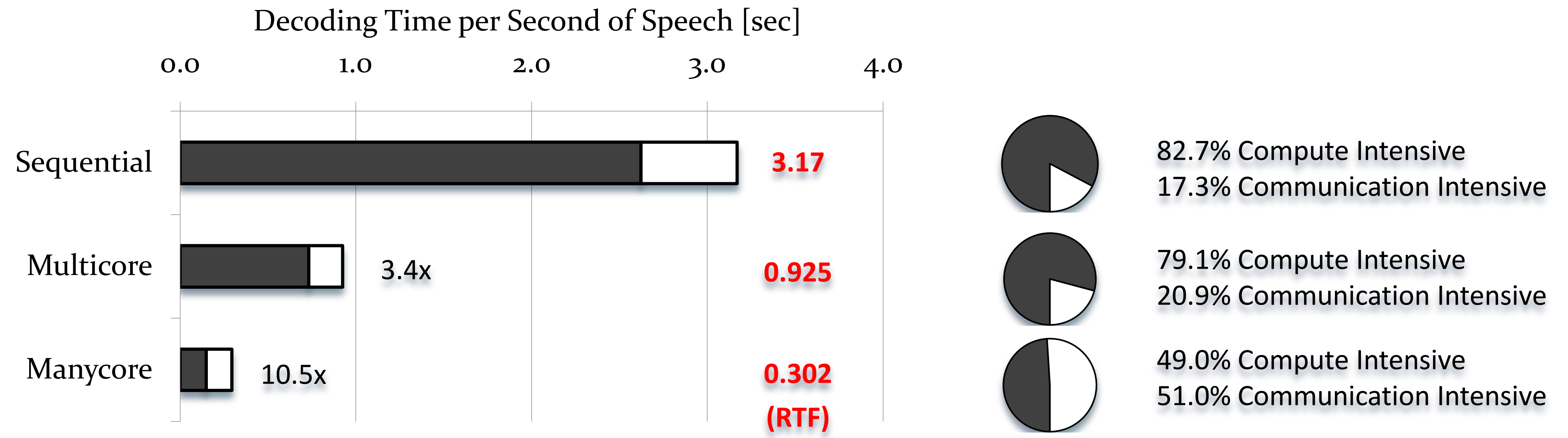
- Principal Architect, Parasians LLC
  - Help clients in compute-intensive industries **achieve revolutionary performance** on applications directly affecting revenue/cost **with highly parallel computing platforms**
  - Sample project: deployed speech inference engine for call centers analytics
- Ph.D. Researcher, University of California, Berkeley
  - Focusing on **speech recognition** and computational finance
  - Built an application framework that allows speech researchers to effectively develop applications on systems with HW accelerators
- Relevant prior experience:
  - Sun Microsystems Inc: Micro-architecture design of the flagship T2 processor
  - Intel, Corp: Optimization of kernels on new MIC processors
  - Xilinx, Inc: Design of application specific multi-ported memory controller

Automatic Speech Recognition



- Allows multimedia content to be transcribed from acoustic waveforms to word sequences
- Emerging commercial usage scenarios in customer call centers
  - Search recorded content
  - Track service quality
  - Provide early detection of service issues

Accelerating Speech Recognition



Kisun You, Jike Chong, *et al*, "Parallel Scalability in Speech Recognition: Inference engine in large vocabulary continuous speech recognition", IEEE Signal Processing Magazine, vol. 26, no. 6, pp. 124-135, November 2009.

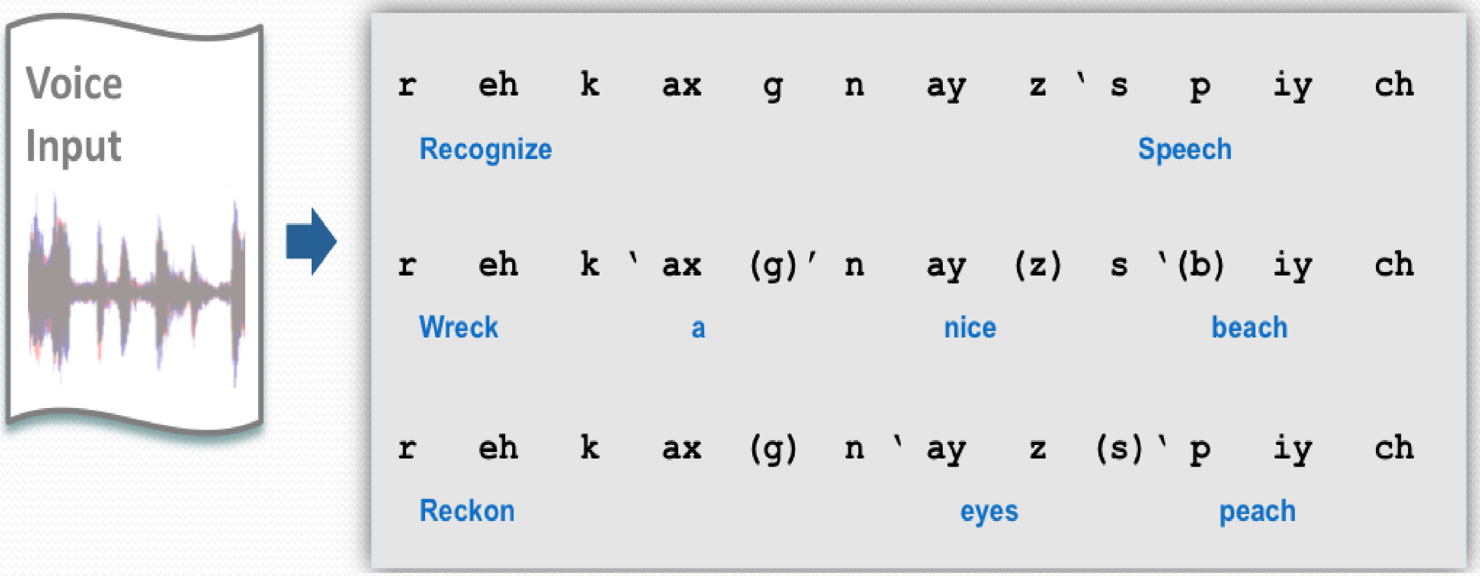
- Demonstrated that speech recognition is amenable to acceleration
  - Fastest algorithm style differed for each platform

Both ***application domain expertise*** and ***hardware architecture expertise*** required to fully exploit acceleration opportunities in an application

Automatic Speech Recognition

- Speech Application Characteristics
  - Typical input/output data types
  - Working set sizes
  - Modules and their inter-dependences
- Four Parallelization Opportunities
  - Over speech segments
  - Over Viterbi forward/backward pass
  - Over phases in each time step
  - Over alternative interpretations
- Four Challenges and Solutions for Efficient GPU Implementation
  - Handling irregular graph structure
  - Efficiently implementing “memoization”
  - Implementing conflict free reduction
  - Parallel construction of global task queues
- An Application Framework for Domain experts
  - Allowing Java/Matlab programmer to get 20x speedup using GPUs

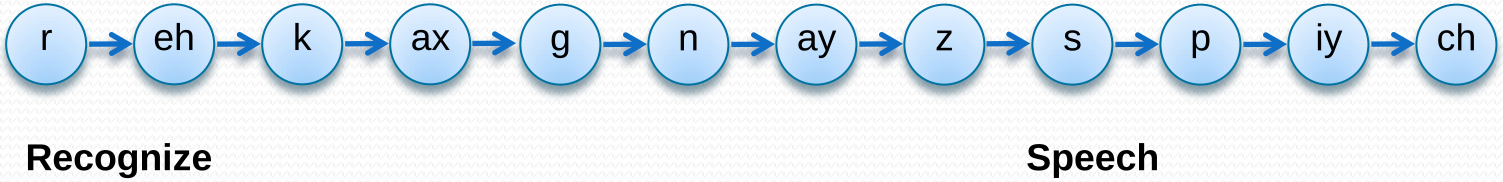
Automatic Speech Recognition



- Challenges:
  - Recognizing words from a large vocabulary arranged in exponentially many possible permutations
  - Inferring word boundaries from the context of neighboring words
- Hidden Markov Model (HMM) based approach is the most successful

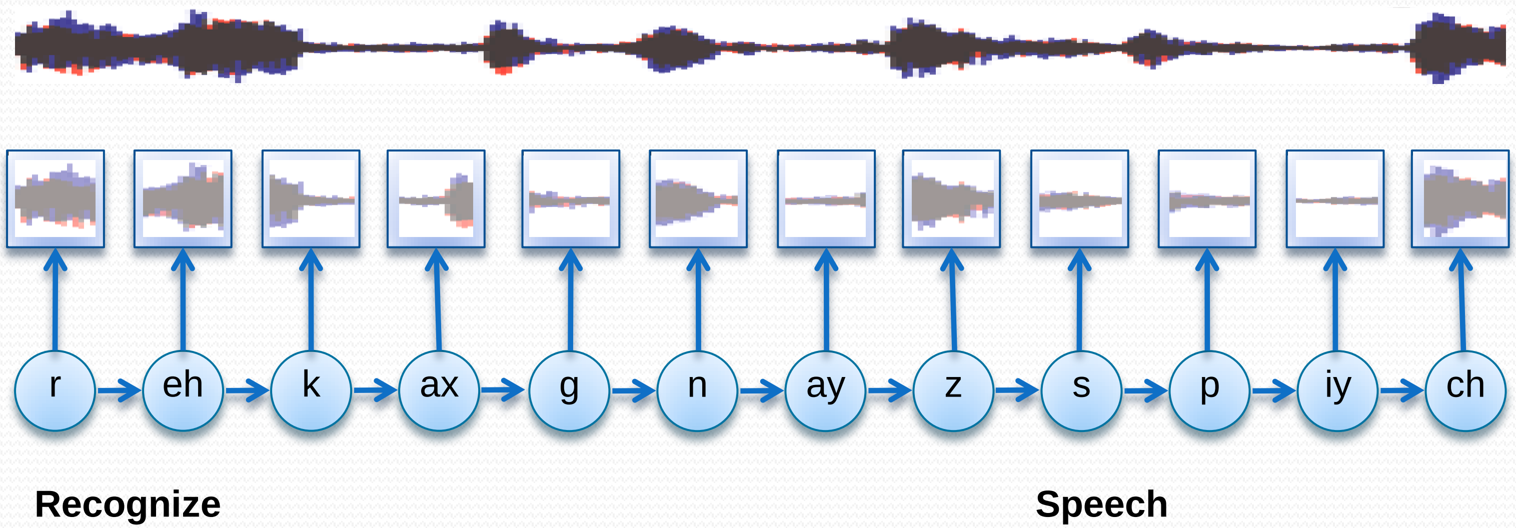
Automatic Speech Recognition

- The Hidden Markov Model approach views *utterances* as following the ***Markov Process***
  - *Utterances* are sequences of phones produced by a speaker
  - ***Markov Process*** describes *sequence* of possibly dependent random variables where any prediction of the next value (x_n), is based on (x_{n-1}) alone
    - Sometime described as a *memoryless* model
    - Flexibly represents any utterances that can be said
    - Use discrete random variables to represent the states in models of languages

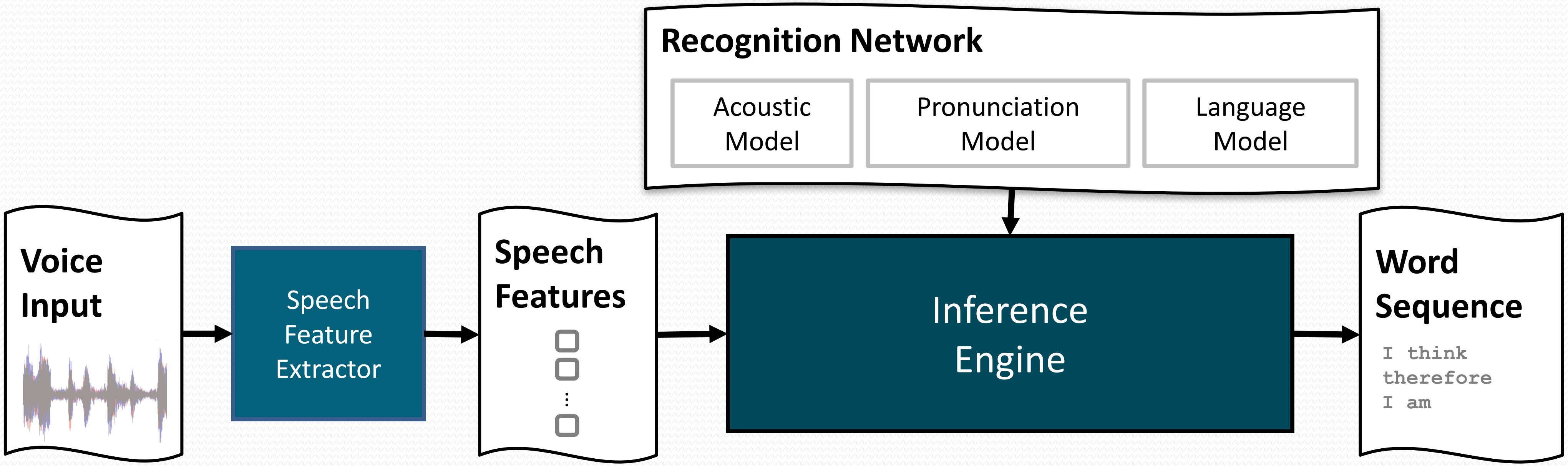


Automatic Speech Recognition

- In the Hidden Markov Model, states are ***hidden***, because phones are ***indirectly observed***
- One must infer the ***most likely interpretation*** of the waveform while taking the model of the ***underlying language*** into account

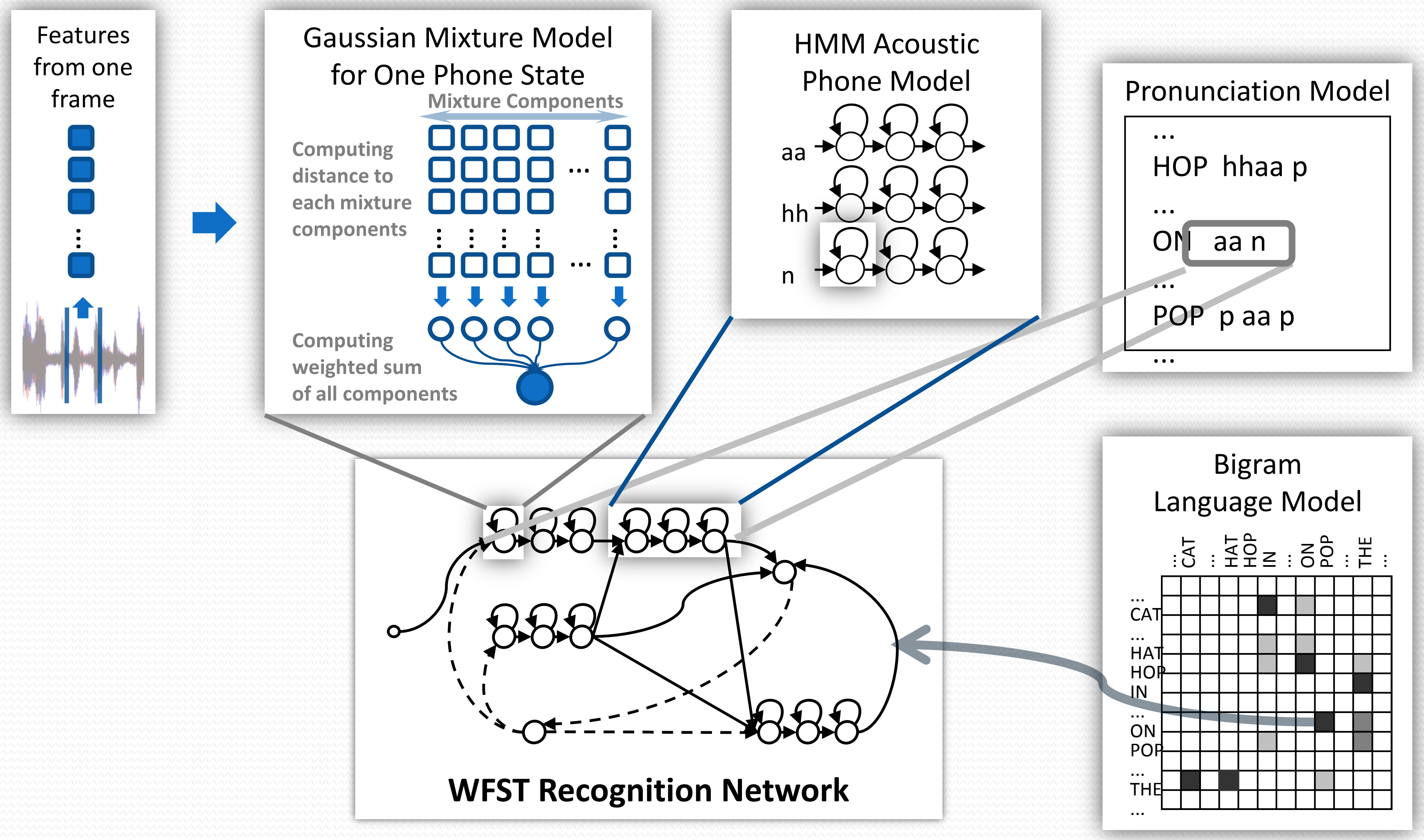


Detailed Algorithm



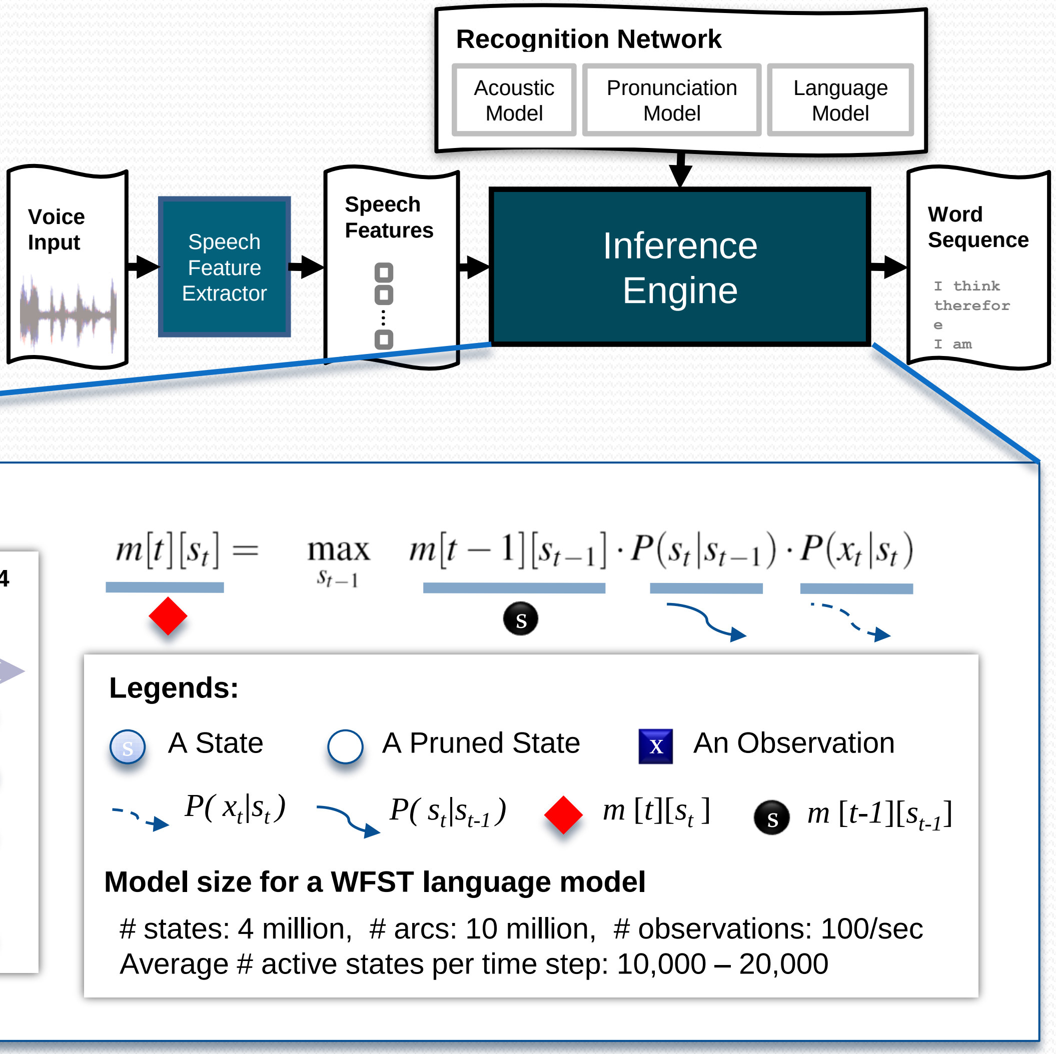
- Inference engine based system
  - Used in Sphinx (CMU, USA), HTK (Cambridge, UK), and Julius (CSRC, Japan)
- Modular and flexible setup
  - Shown to be effective for Arabic, English, Japanese, and Mandarin

Detailed Algorithm



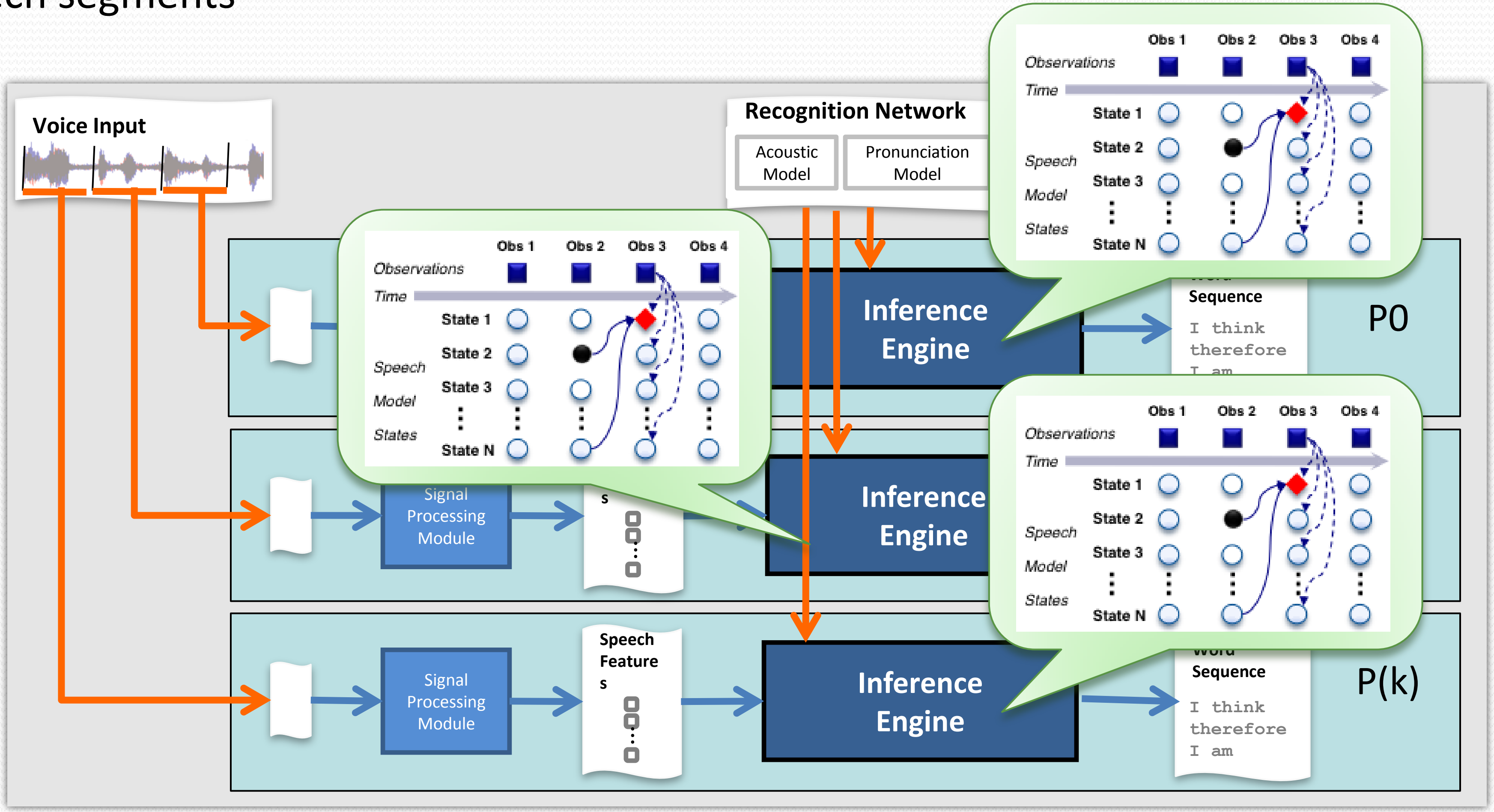
Application Context

- Speech inference uses the Viterbi algorithm
  - Evaluate one observation at a time
    - based on 10ms window of acoustic waveform
  - Computing the state with three components
    - Observation probability
    - Transition probability
    - Prior likelihood



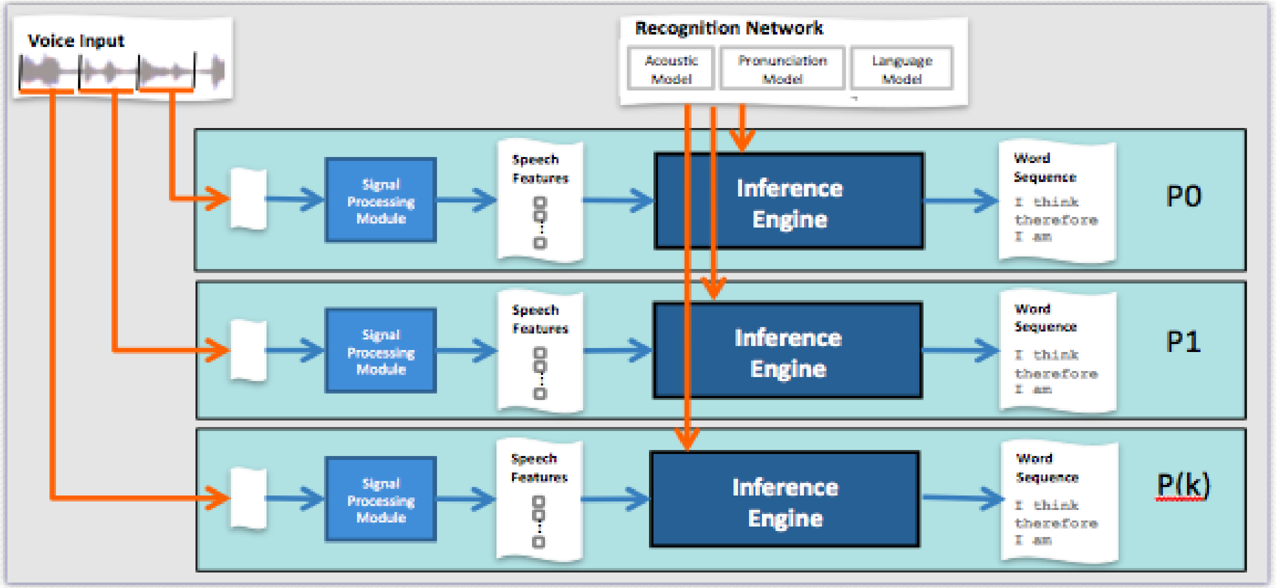
Acceleration Opportunity (1)

- Concurrency over speech segments
  - Multiple inference engine working on different segments of speech
  - Shared recognition network



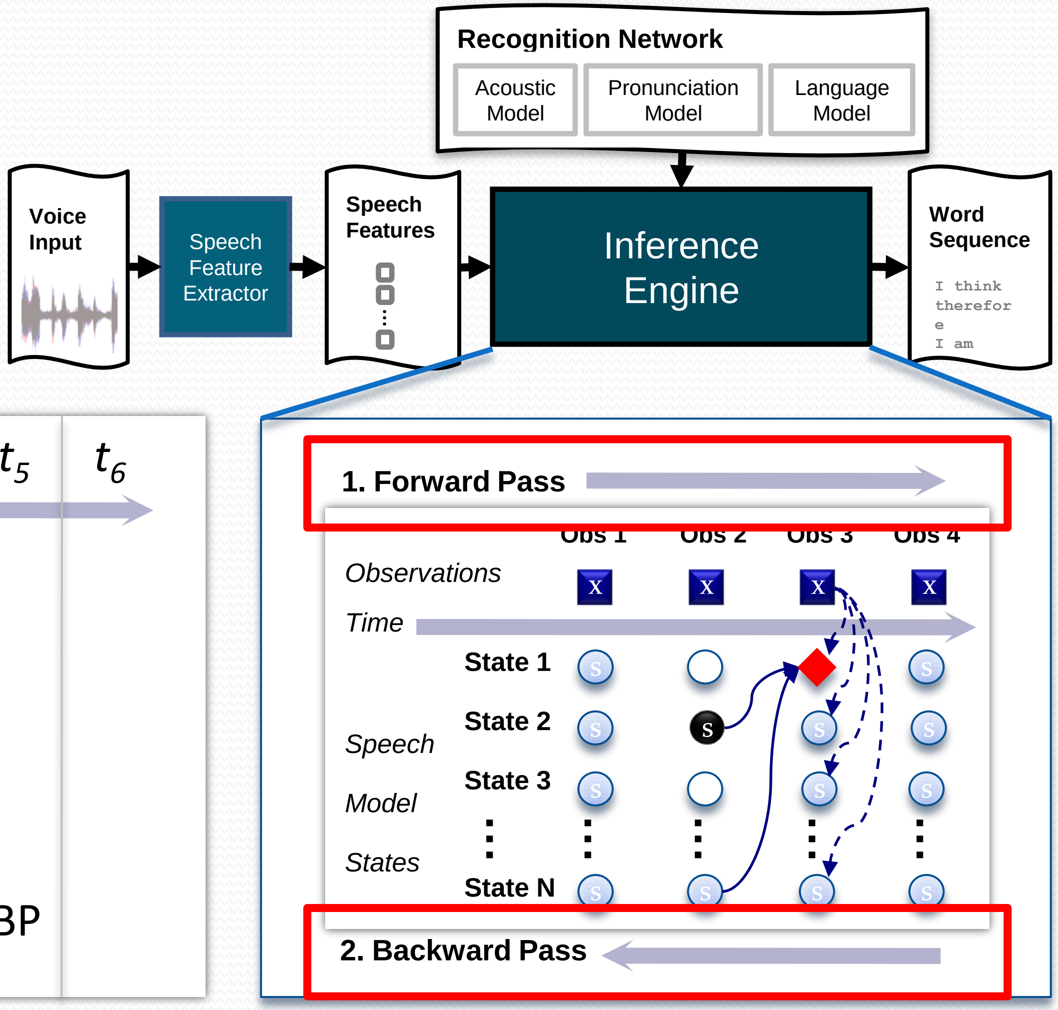
Mapping Concurrency to GPUs (1)

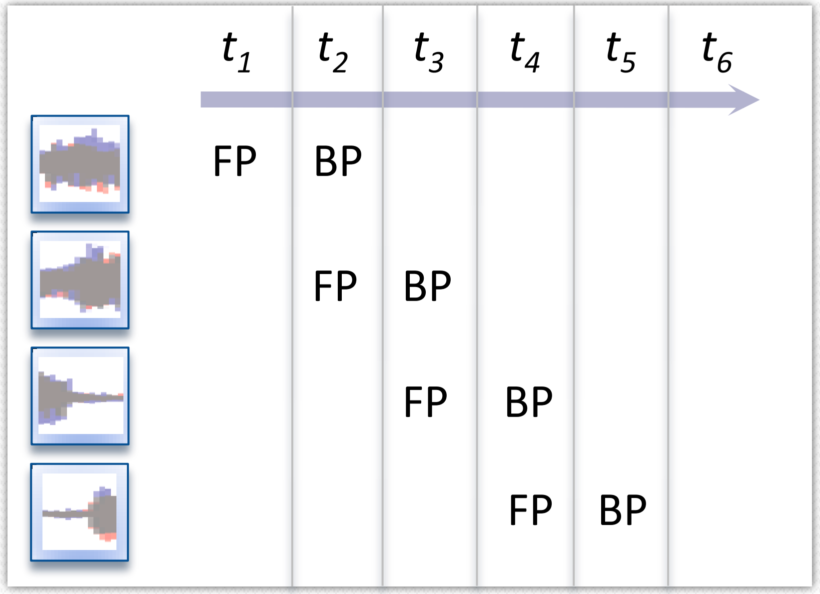
- Concurrency among speech utterances is the low hanging fruit
  - Can be exploited over multiple processors
  - Complementary to the more challenging fine-grained concurrency
- However, exploiting it among cores and vector lanes is ***not practical***
  - Local scratch-space not big enough
  - Access to global memory is shared
  - Significant memory bandwidth required



Acceleration Opportunity (2)

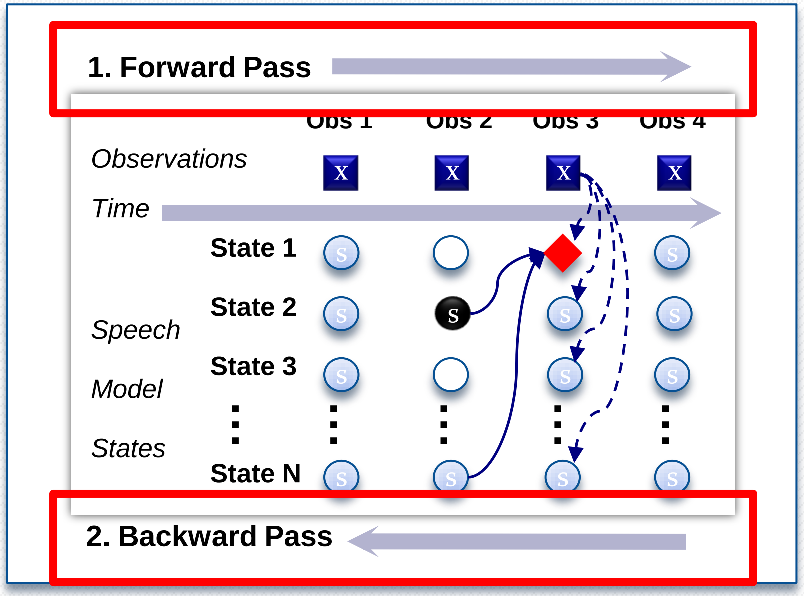
- Concurrency over forward pass and backward pass of the Viterbi algorithm
  - Pipelining two parts of the algorithm by operating on different segments of speech





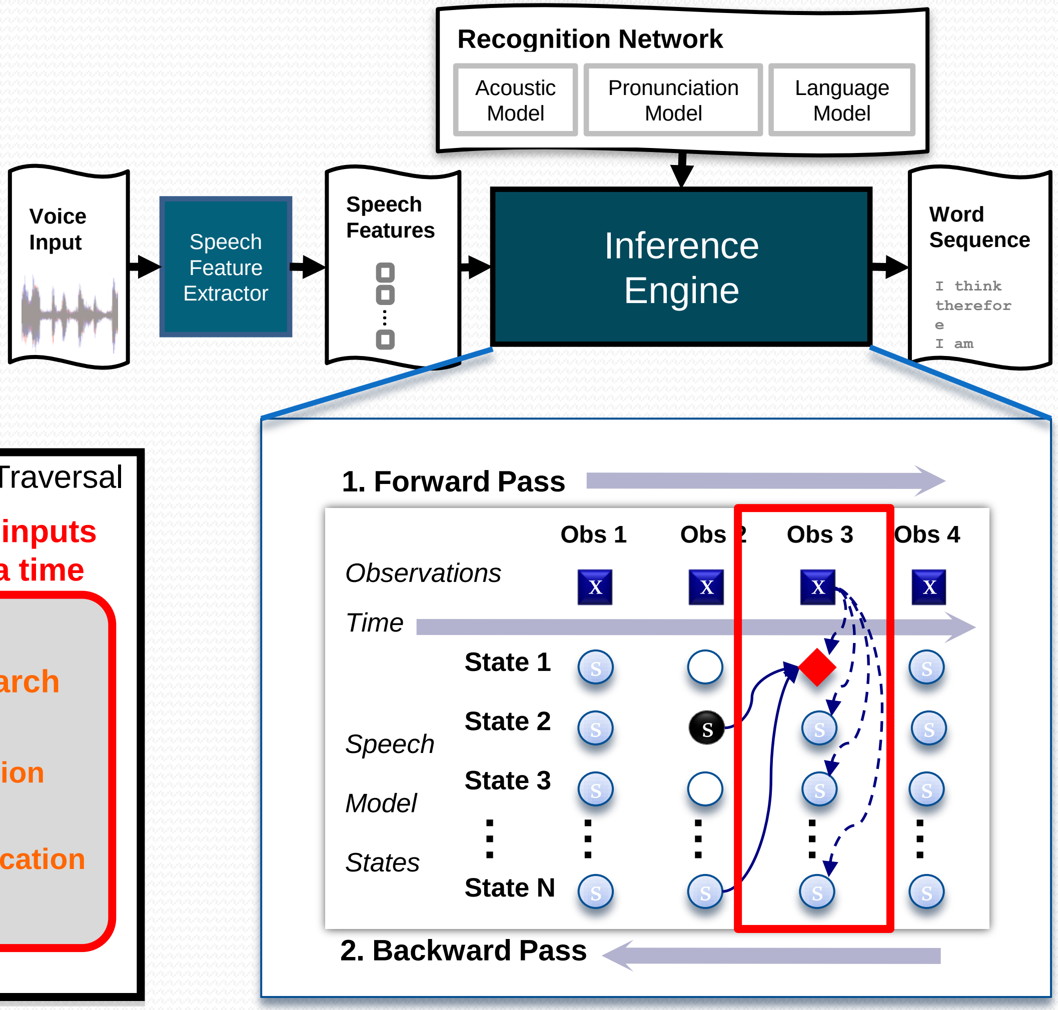
Mapping Concurrency to GPUs (2)

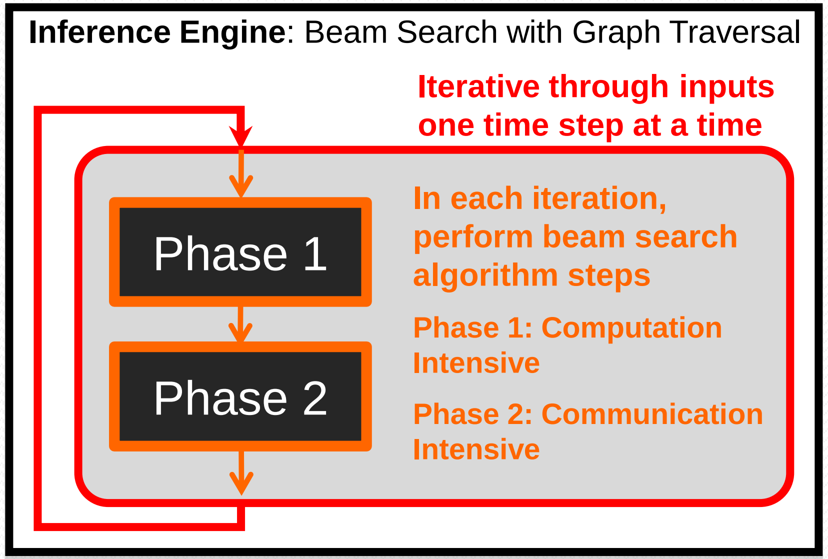
- Concurrency among forward and backward passes is exploitable
  - To effectively pipeline, stages should be balanced
- Forward Pass: >99% of execution time
- Backward Pass: <1% of execution time
- Exploiting it will not result in appreciable performance gain



Acceleration Opportunity (3)

- Concurrency over each algorithm phase in the forward pass of each time step
  - Phase 1: compute intensive
  - Phase 2: communication intensive

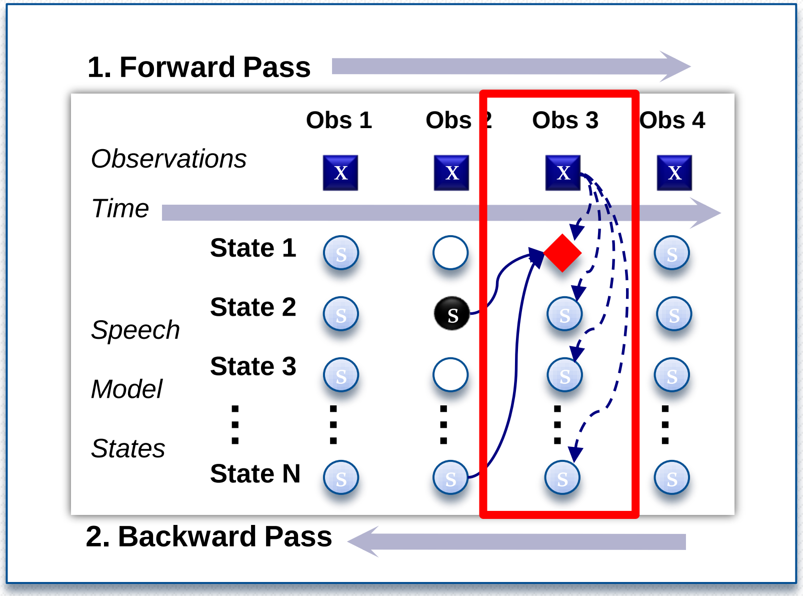




Mapping Concurrency to GPUs (3)

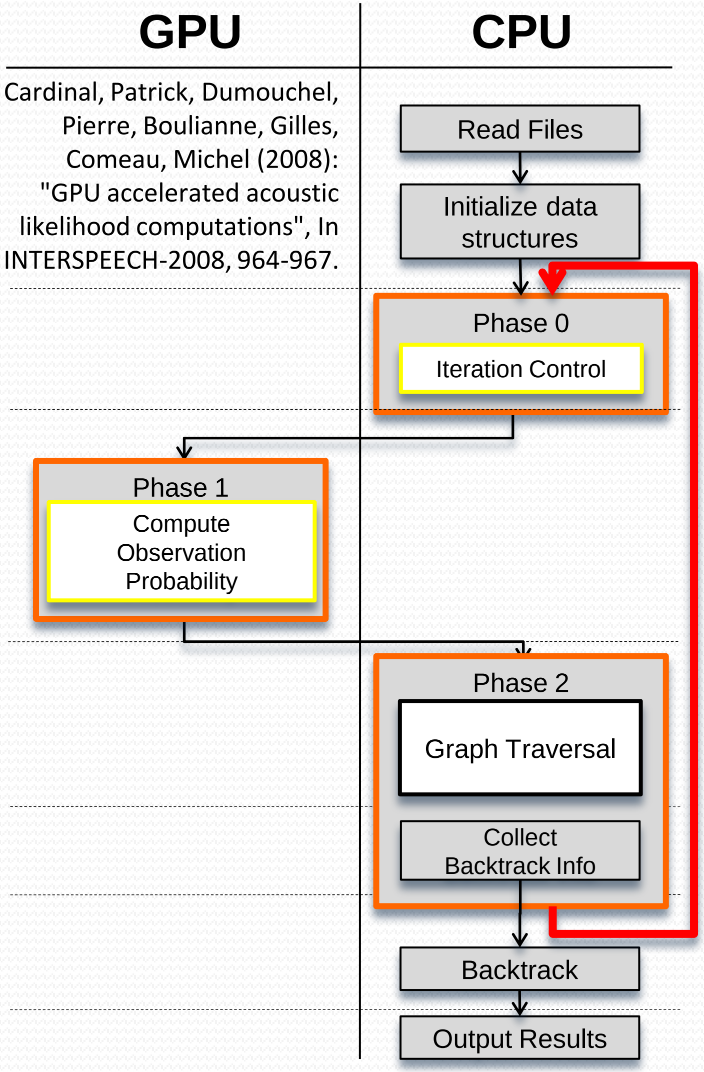
- Concurrency among Phase 1 (compute intensive phase) and Phase 2 (communication intensive phase) is exploitable
  - In the parallelized version, the two phases have similar execution times

- However, transferring data between the two phases may be a bottleneck
  - Bottleneck observed when
    - Phase 1 ➔ (CPU)? (GPU)?
    - Phase 2 ➔ (CPU)? (GPU)?

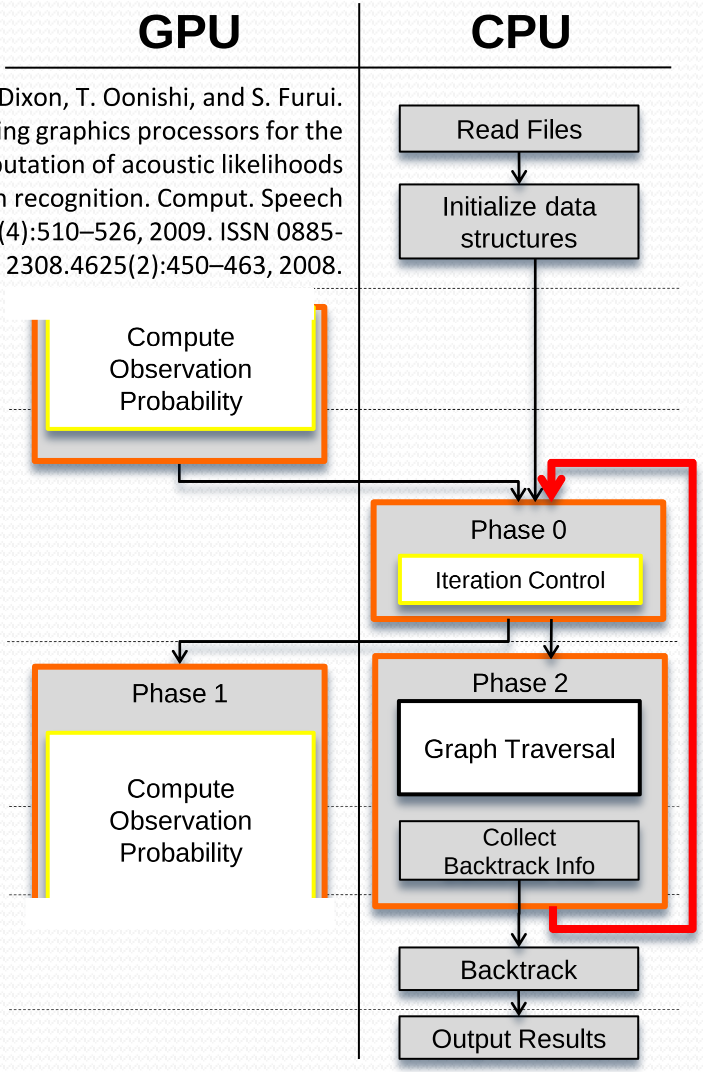


Mapping Concurrency to GPUs (3)

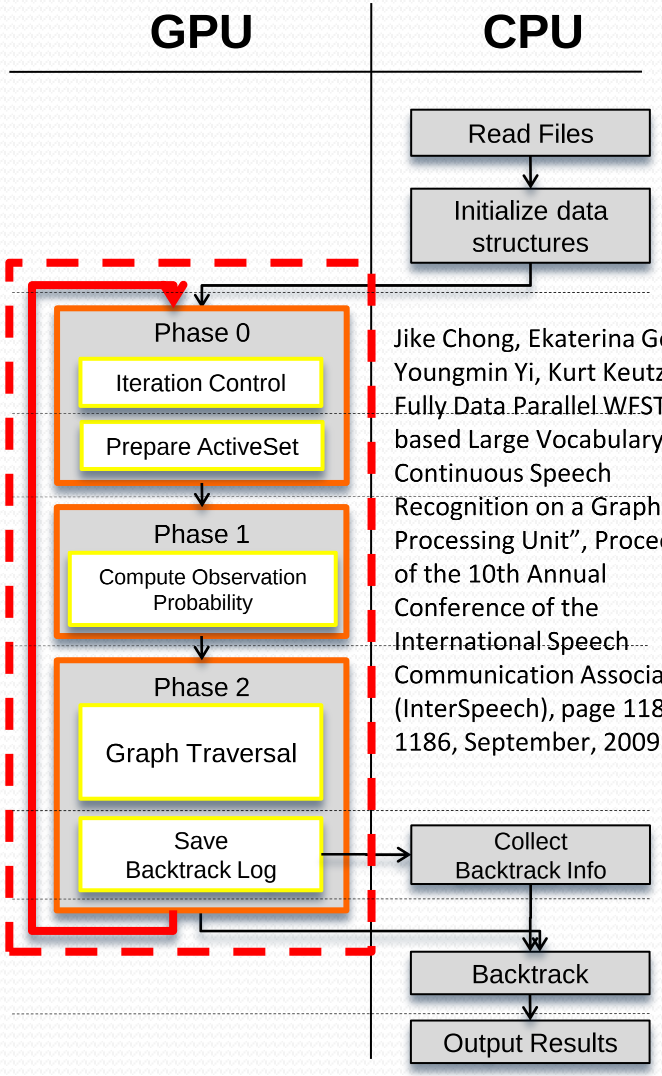
Cardinal, Patrick, Dumouchel, Pierre, Boulianne, Gilles, Comeau, Michel (2008): "GPU accelerated acoustic likelihood computations", In INTERSPEECH-2008, 964-967.



P. R. Dixon, T. Oonishi, and S. Furui. Harnessing graphics processors for the fast computation of acoustic likelihoods in speech recognition. Comput. Speech Lang., 23(4):510–526, 2009. ISSN 0885-2308.4625(2):450–463, 2008.

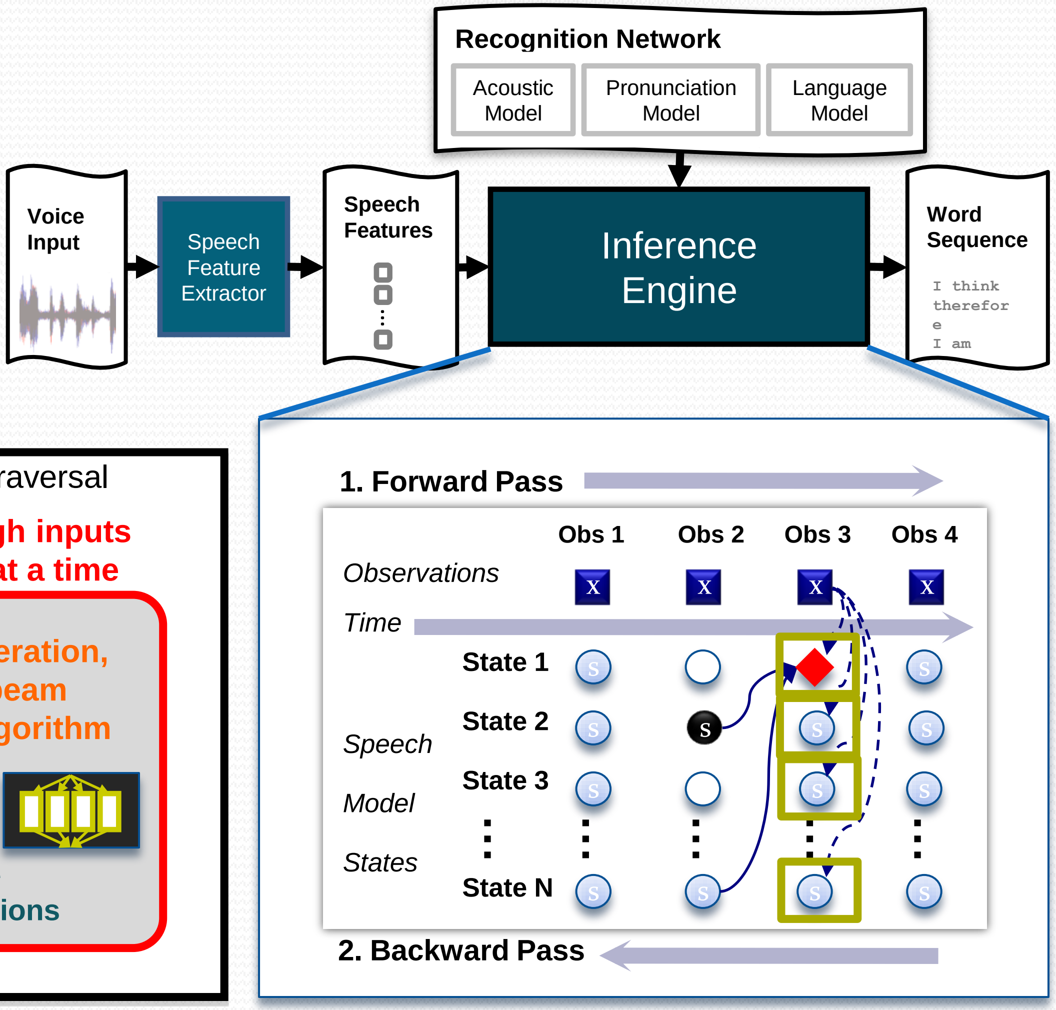


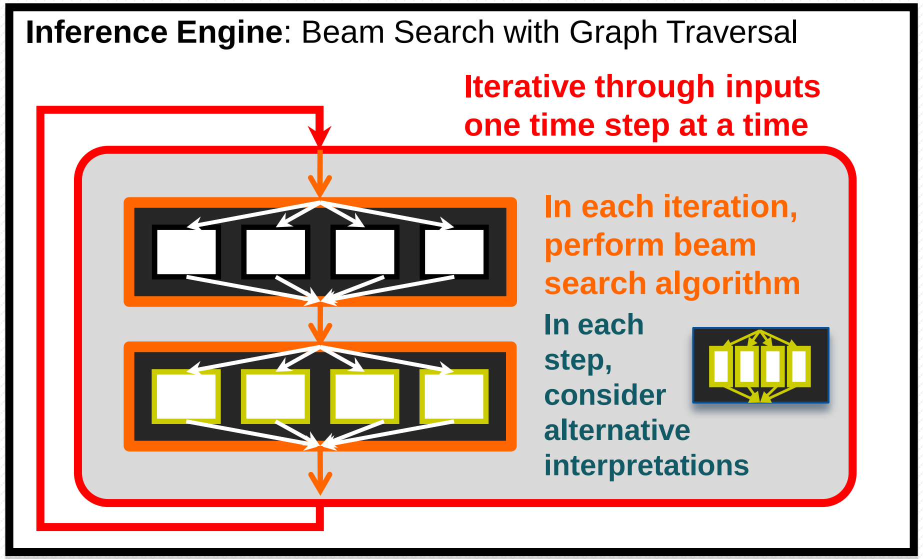
Jike Chong, Ekaterina Gonina, Youngmin Yi, Kurt Keutzer, "A Fully Data Parallel WFST-based Large Vocabulary Continuous Speech Recognition on a Graphics Processing Unit", Proceeding of the 10th Annual Conference of the International Speech Communication Association (InterSpeech), page 1183 – 1186, September, 2009.



Acceleration Opportunity (4)

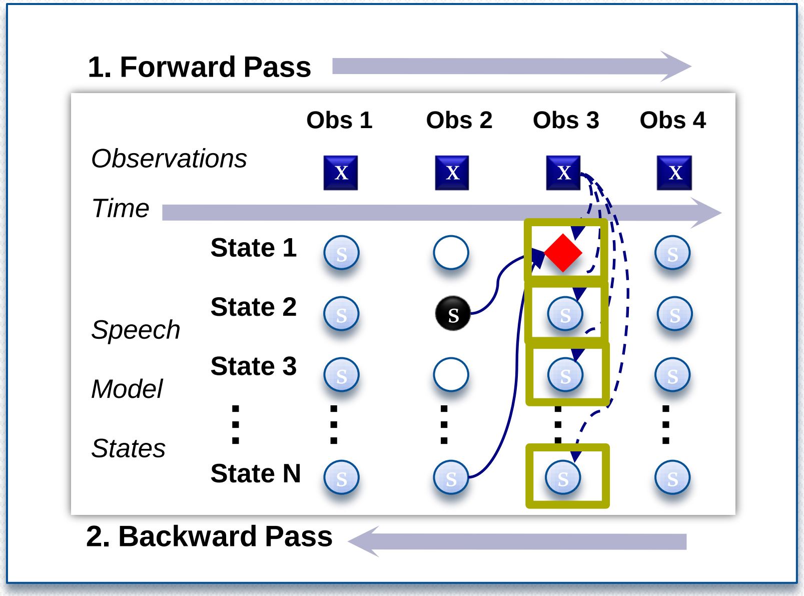
- Concurrency over alternative interpretations of the utterance within each algorithm step
  - Computing the state with three components
    - Observation probability
    - Transition probability
    - Prior likelihood





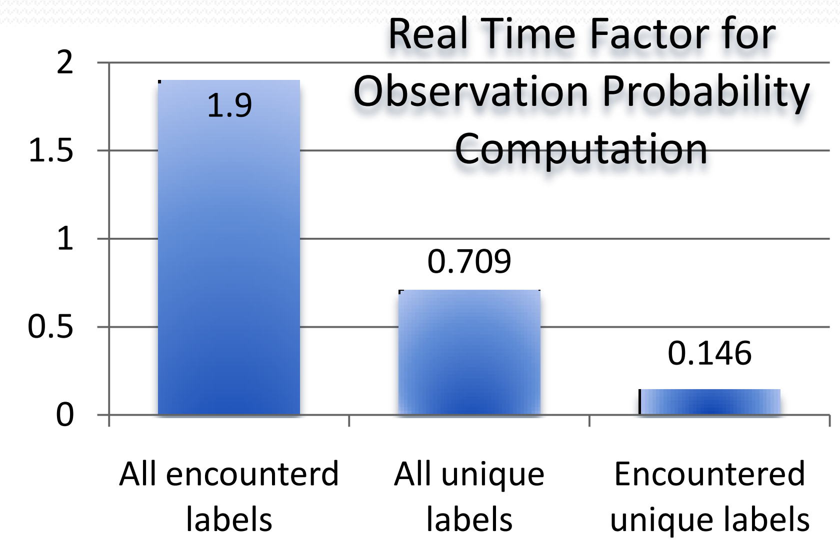
Mapping Concurrency to GPUs (4)

- Concurrency among alternative interpretations is abundant
  - 10,000s alternative interpretations tracked per time step
  - Well matched to the architecture of the GPU
- With the concurrency, comes many challenges…
  1. **Eliminating redundant work by implementing parallel "memoization"**
  2. **Handling irregular graph structures with data parallel operations**
  3. **Conflict-free reduction in graph traversal to resolve write-conflicts**
  4. **Parallel construction of a task queue while avoiding sequential bottlenecks at queue control variables**

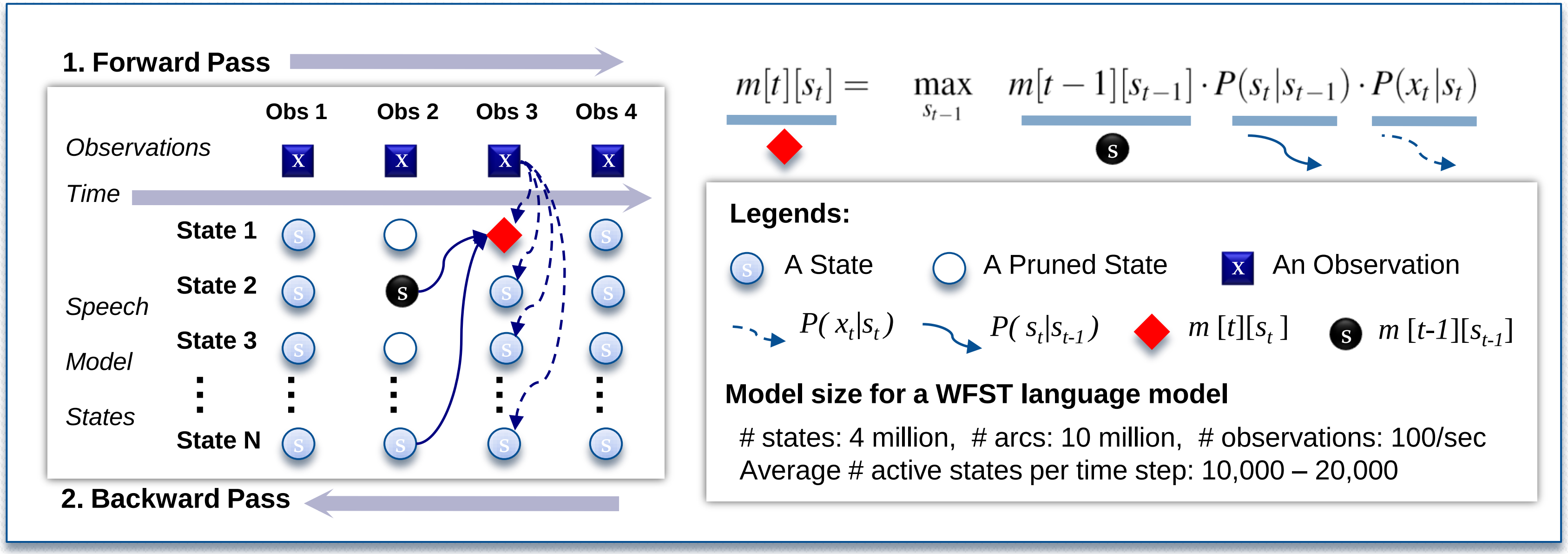


Challenge 1:

- Eliminating redundant work when threads are computing results for an unpredictable subset of the problems based on input
  - Only 20% of the triphone states are used for observation probability computation
  - Many duplicate labels
  - In sequential execution, memoization is used to avoid redundancy
  - What do we do on data-parallel platforms?

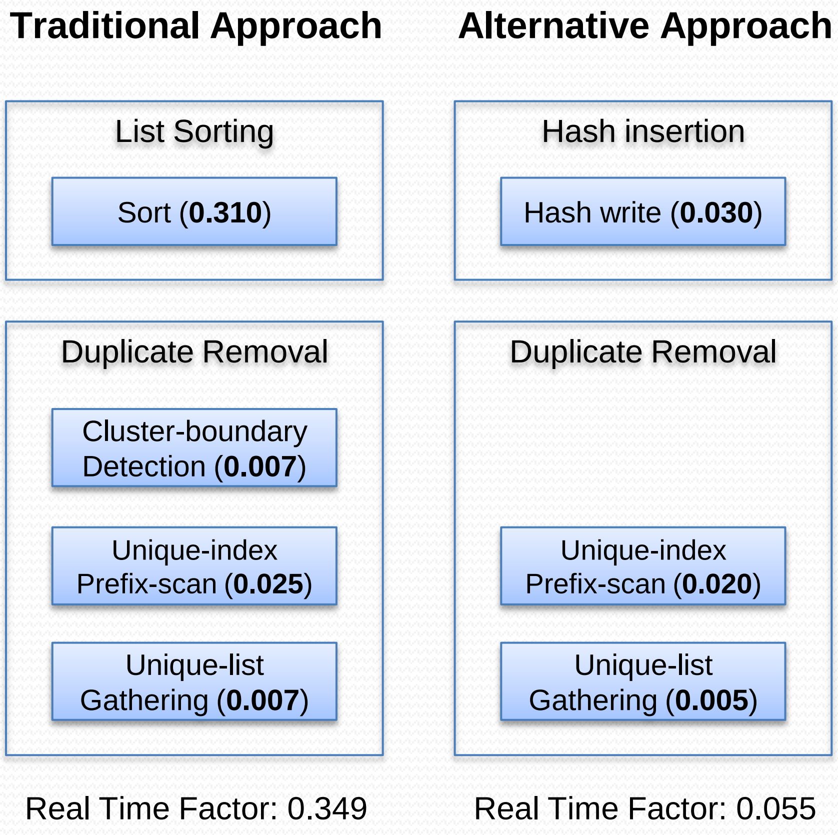


Real Time Factor shows the number of seconds required to process one second of input data



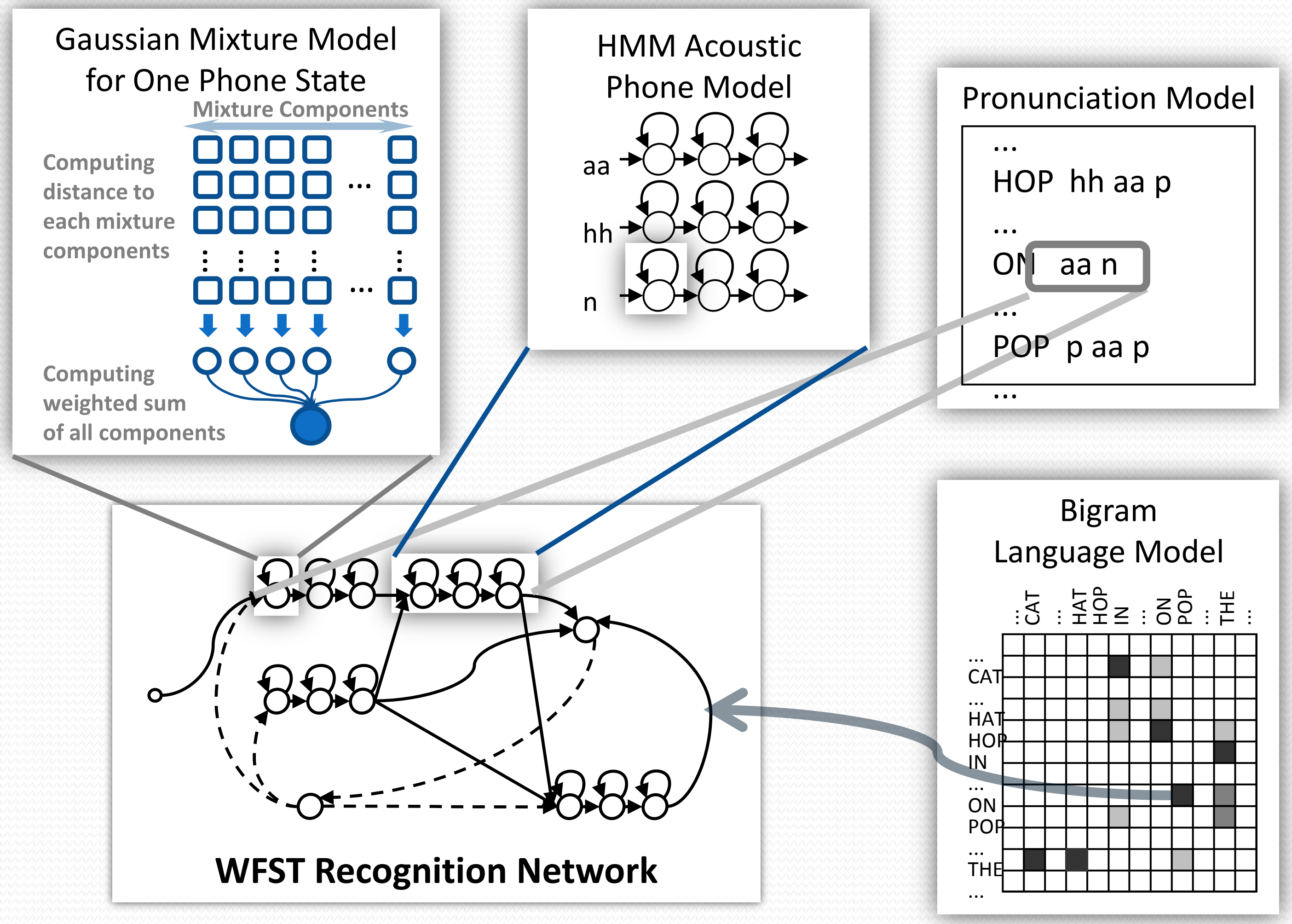
Solution 1:

- Implement efficient find-unique function by leveraging the GPU global memory write-conflict-resolution policy
  - Leverage the semantics of conflicting non-atomic write to use the hash table as a flag array
  - CUDA guarantees at least one conflicting write to a device memory location to be successful, which is enough to build a flag array
  - The alternative “Hash Insertion” step greatly simplifies the find-unique operation



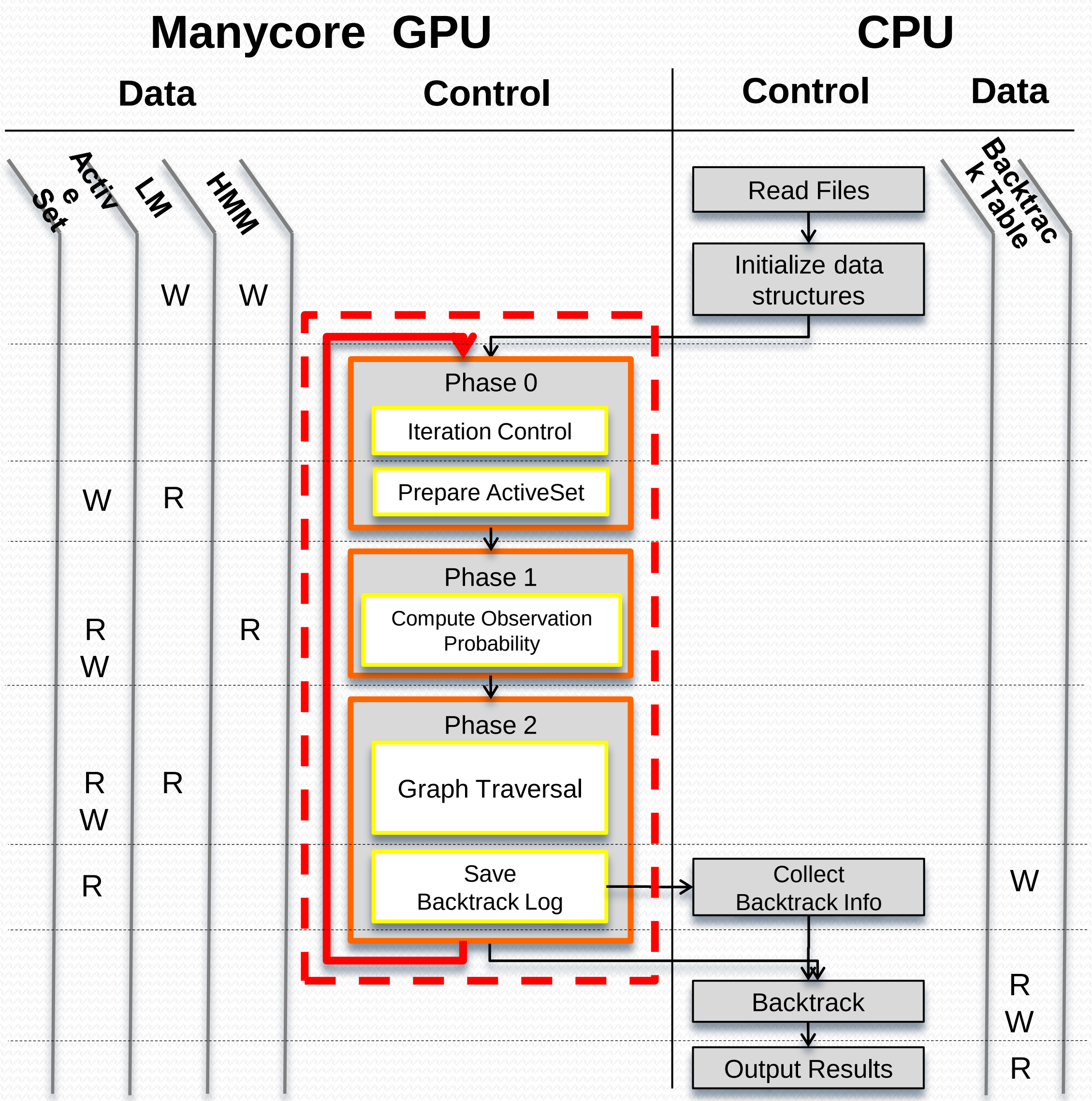
Challenge 2:

- Handling irregular data structures with data-parallel operations
  - Forward pass: ***1,000s to 10,000s*** of concurrent tasks represent most likely ***alternative interpretations*** of the input being tracked
  - To track: reference selected ***subset*** of a ***sparse irregular graph*** structure
  - The concurrent access of irregular data structure requires "***uncoalesced***" memory accesses in the middle of important algorithm steps, which ***degrades performance***



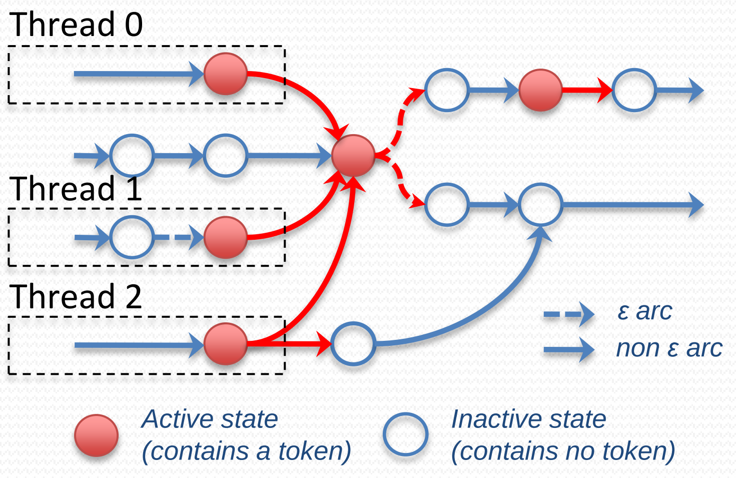
Solution 2:

- Construct efficient dynamic vector data structure to handle irregular data accesses
  - Instantiate a ***Phase 0*** in the implementation to gather all operands necessary for the current time step of the algorithm
  - Caching them in a memory-coalesced runtime data structure allows any uncoalesced accesses to happen only once for each time step



Challenge 3:

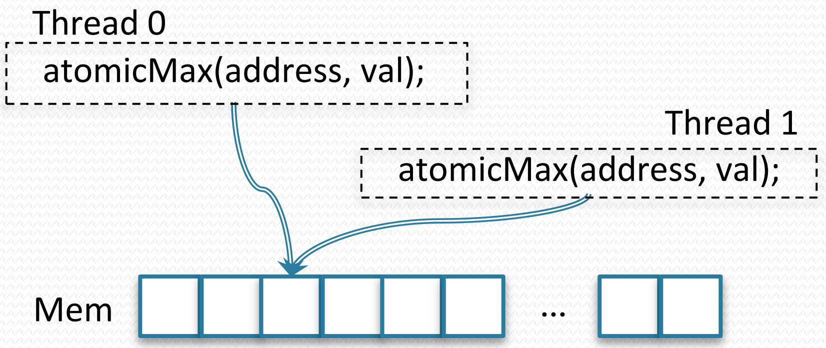
- Conflict-free reduction in graph traversal to implement the Viterbi beam-search algorithm
  - During graph traversal, active states are being processed by parallel threads on different cores
  - Write-conflicts frequently arise when threads are trying to update the same destination states
- To further complicate things, in statistical inference, we would like to only keep the most likely result
  - Efficiently resolving these write conflicts while keeping just the most likely result for each state is essential for achieving good performance



A section of a
Weighed Finite State Transducer Network

Solution 3:

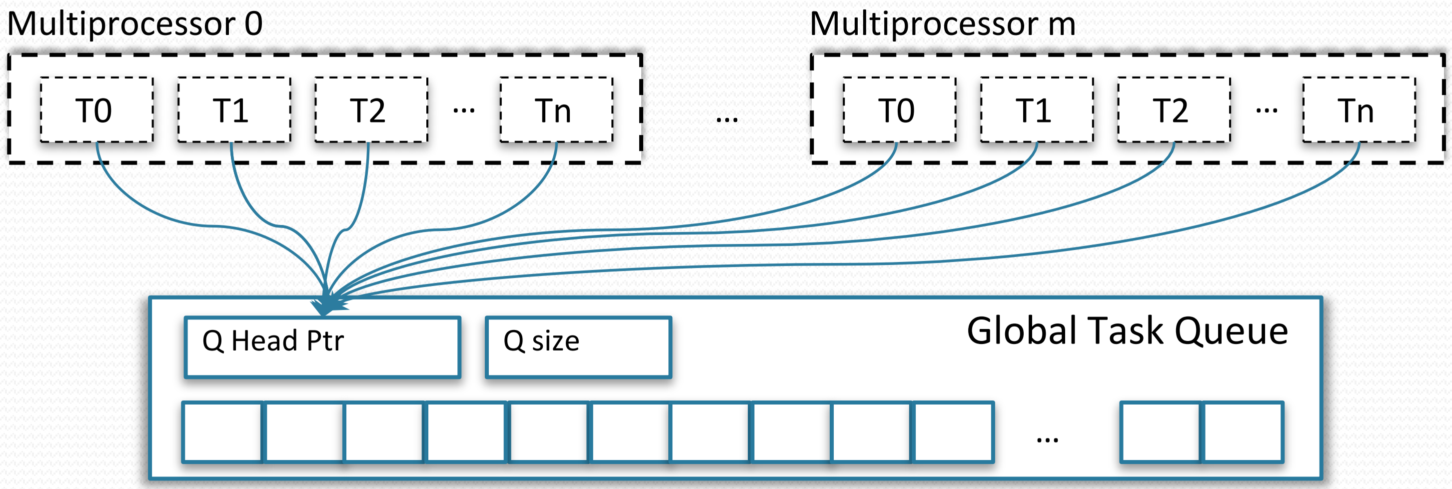
- Implement lock-free accesses of a shared map leveraging advanced GPU atomic operations to enable conflict-free reductions
  - CUDA offers atomic operations with various flavors of arithmetic operations
  - The “atomicMax” operation is ideal for statistical inference
  - Final result in each atomically accessed memory location will be the maximum of all results that was attempted to be written to that memory location
  - This type of access is lock-free from the software perspective, as the write-conflict resolution is performed by hardware
  - Atomically writing results in to a memory location is a process of reduction, hence, this is a ***conflict-free reduction***




```
int   stateID       = ActiveStateIDList[tid];
float res           = compute_result( tid );
int   valueAtState =
         atomicMax(&(destStateProb[ stateID ]), res);
```

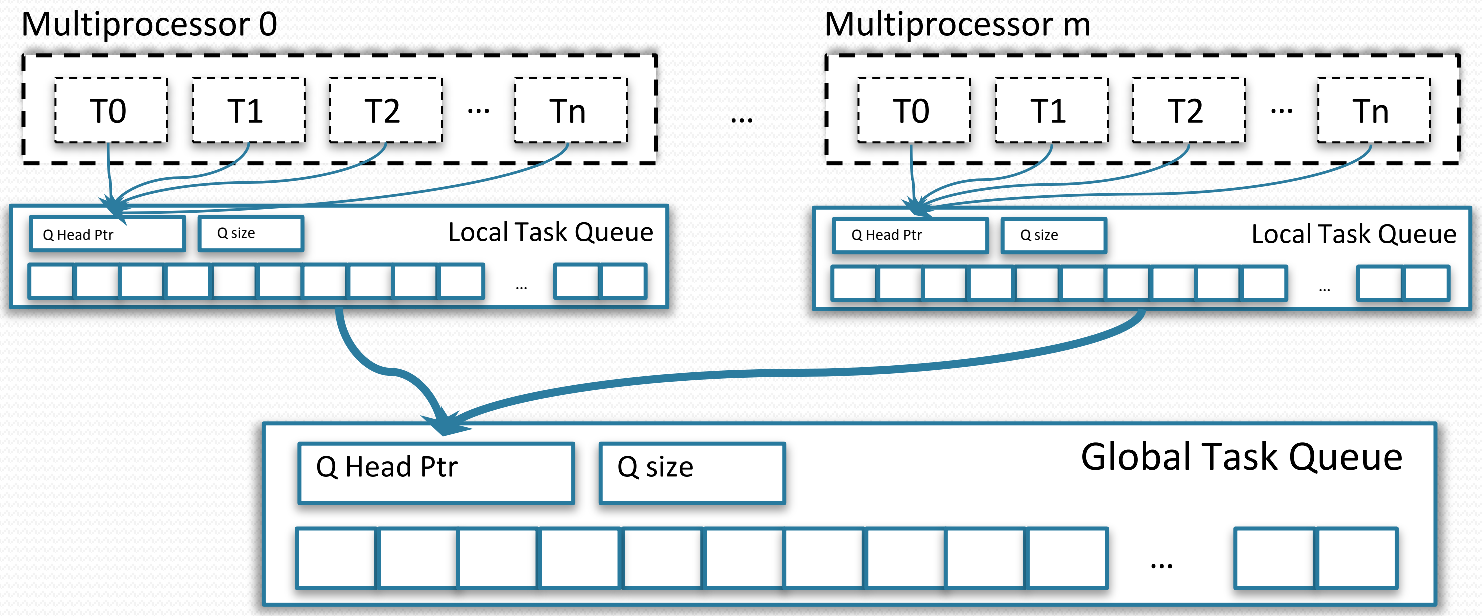
Challenge 4: Global Queue Contention

- 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 serialization occurs at the point of the queue control variables



Solution 4: Hybrid Global/Local Queue

- 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, the single point of serialization is eliminate
  - Each multiprocessor can build up its local queue using local atomic operations, which have lower latency than the global atomic operations
  - The writes to the shared global queue are performed in one batch process, and thus are significantly more efficient



Solution 4: Hybrid Global/Local Queue

```
// Local Q: shared memory data structure
// ----------------------------------------------------
extern shared int sh_mem[];
int *myQ = (int *) sh_mem;          // memory for local Q
shared int myQ_head, globalQ_index; // Queue Ctrl Variables
if(threadIdx.x==0){ myQ_head = 0;}  syncthreads();

// Constructing the queue content in Local Q
// ----------------------------------------------------
int tid = blockIdx.x*blockDim.x + threadIdx.x;
if(tid<nStates) {
  int stateID = ActiveStateIDList[tid];
  float res = compute_result( tid );
  if (res < pruneThreshold) {res = FLTMIN;}
  else {
    //if res is more likely than threshhold, then keep
    int valueAtState =
        atomicMax(&(destStateProb [ stateID ]) , res );
    // If no duplicate, add to local Q
    if ( valueAtState == INIT_VALUE) {
      int head=atomicAdd(&myQ_head ,1 ) ;
      myQ[ head ] = stateID ;
    }
  }
```

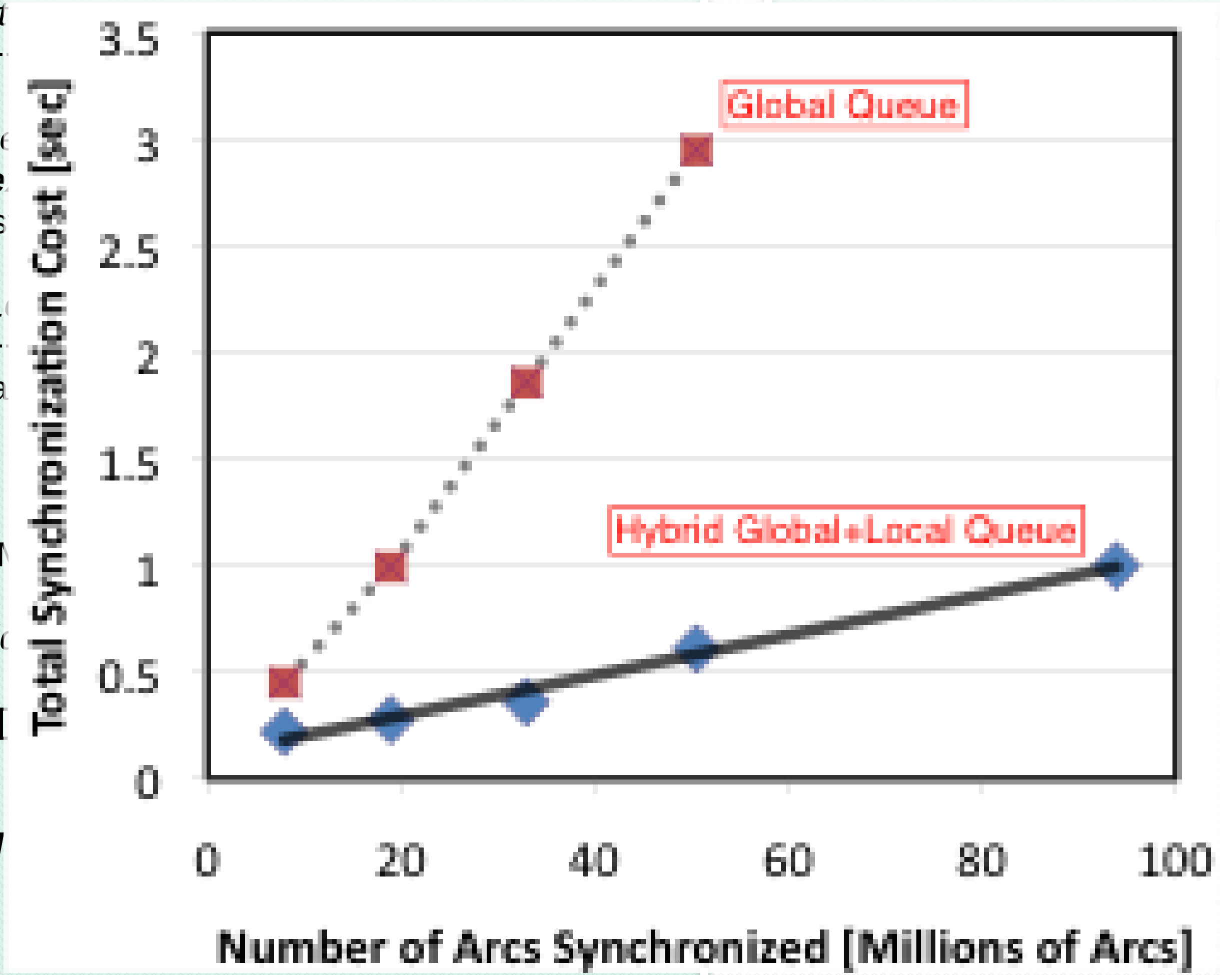
```
  // Local Q –> Global Q transfer
  // --------------------------------------
  syncthreads ();
  if (threadIdx.x==0) {
    globalQ_index =
      atomicAdd(stateHeadPtr , myQ head);
  }
  syncthreads ();
  if (threadIdx.x<myQhead)
    destStateQ [globalQ_index+threadIdx.x] =
        myQ[ threadIdx . x ] ;
} // end if(tid<nStates)
```

Solution 4: Hybrid Global/Local Queue

```
// Local Q: shared memory data struct
// ----------------------------
extern shared int sh_mem[];
int *myQ = (int *) sh_mem;          // me
shared int myQ_head, globalQ_inde
if(threadIdx.x==0){ myQ_head = 0;}  s

// Constructing the queue content in L
// ----------------------------
int tid = blockIdx.x*blockDim.x + threa
if(tid<nStates) {
  int stateID = ActiveStateIDList[tid];
  float res = compute_result( tid );
  if (res < pruneThreshold) {res = FLTM
  else {
    //if res is more likely than threshhol
    int valueAtState =
        atomicMax(&(destStateProb [
    // If no duplicate, add to local Q
    if ( valueAtState == INIT_VALUE) {
      int head=atomicAdd(&myQ_head
      myQ[ head ] = stateID ;
    }
  }
```

```
sfer
-------------
, myQ head);
ex+threadIdx.x] =
```



Four Main Techniques for the GPU

1. Constructing ***efficient dynamic vector data structures*** to handle ***irregular graph traversals***

2. Implementing an ***efficient find-unique function*** to ***eliminate redundant work*** by leveraging the GPU global memory write-conflict-resolution policy

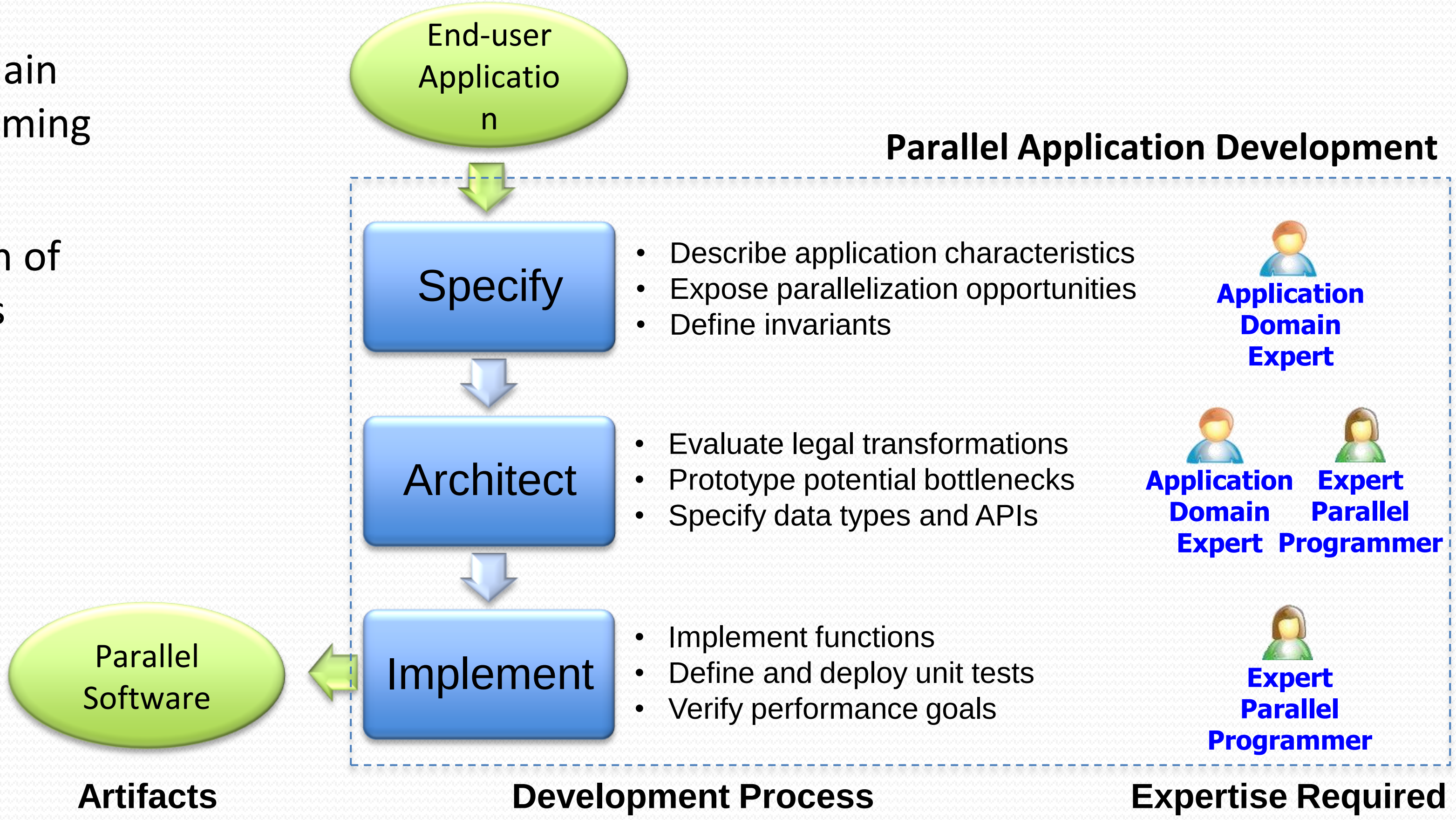
3. Implementing ***lock-free accesses*** of a shared map leveraging advanced GPU atomic operations to enable ***conflict-free reduction***

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

Jike Chong, Ekaterina Gonina, Kurt Keutzer, "Efficient Automatic Speech Recognition on the GPU", accepted book chapter in GPU Computing Gems, Vol. 1.

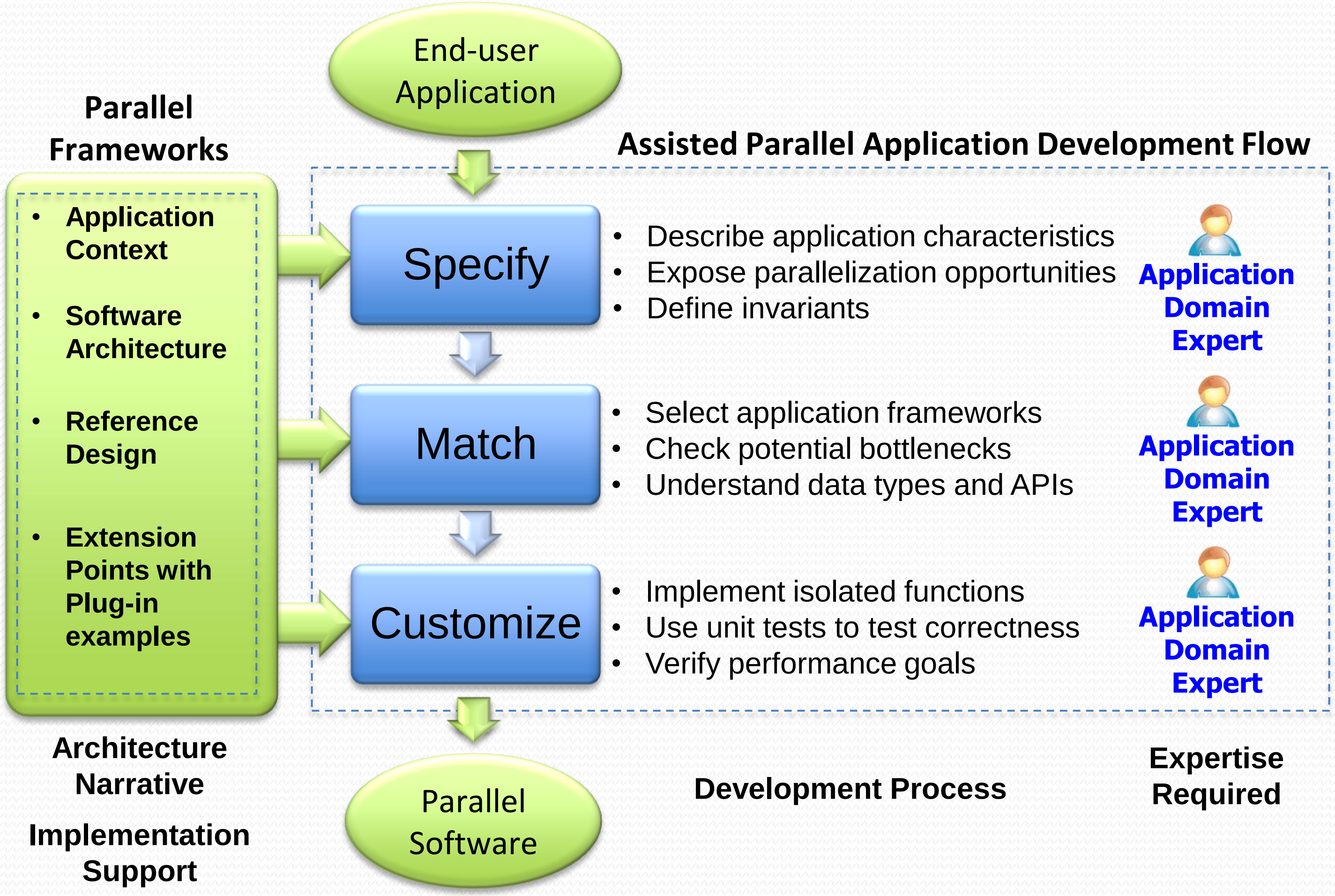
Parallel Software Development

- Industry best practice:
  - Requires both application domain expertise and parallel programming expertise
  - Severely limits the proliferation of highly parallel microprocessors



Parallel Software Development

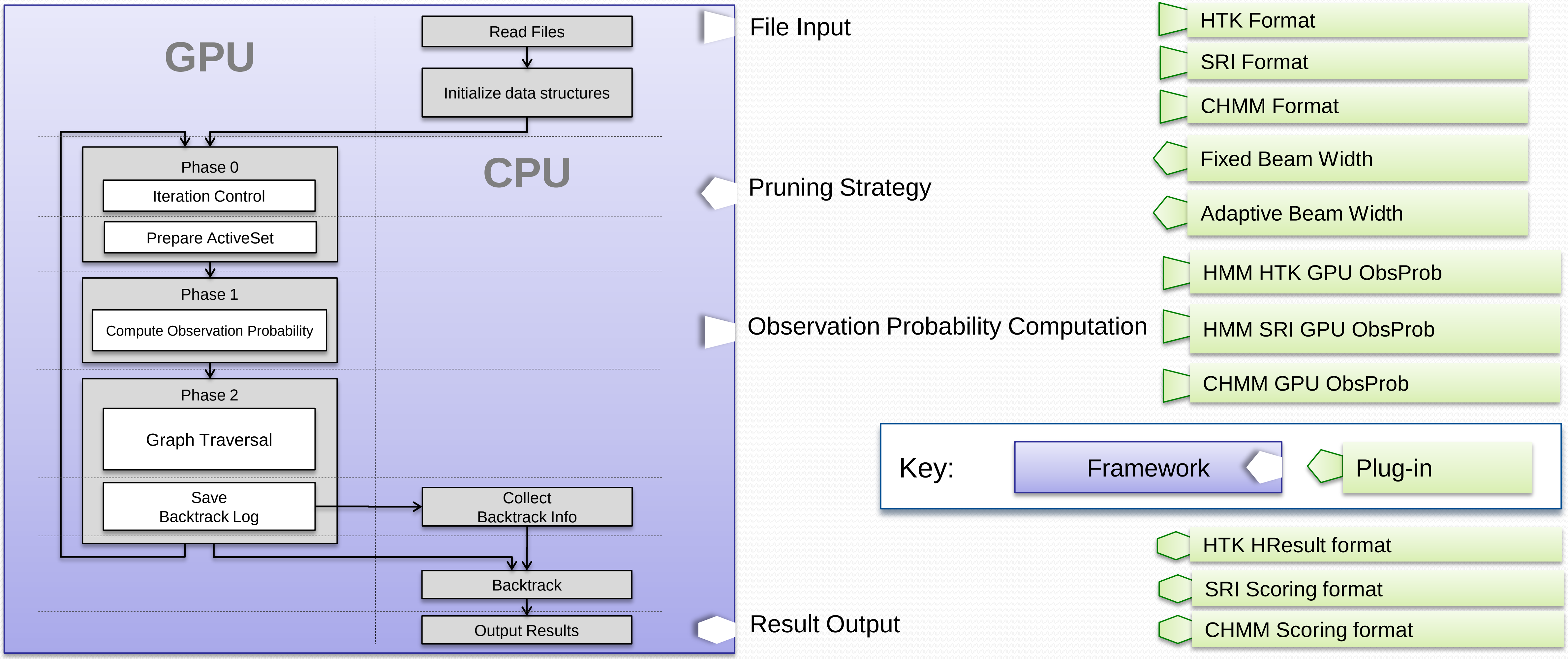
- Industry best practice with assistance from Application Frameworks:
  - Parallel programming expertise encapsulated in application framework
  - Application domain expertise alone is sufficient to utilize highly parallel microprocessors



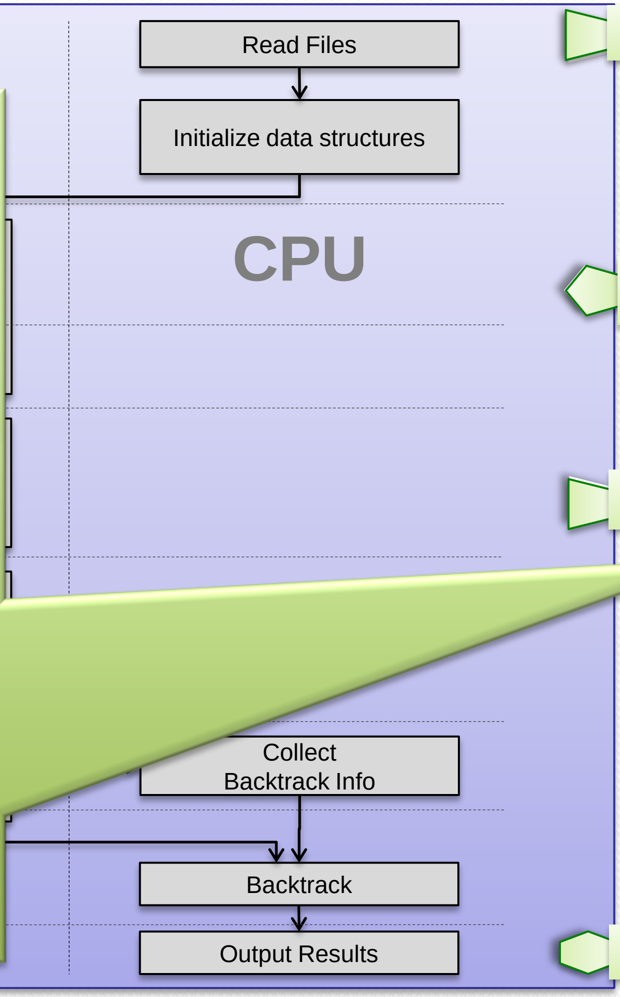
Four Components of an Application Framework

- Application Context
  - A description of the application characteristics and requirements that exposes parallelization opportunities independent of the implementation platform
- Software Architecture
  - A hierarchical composition of parallel programming patterns that assists in navigating the reference implementation
- Reference Implementation
  - A fully functional, efficient sample design of the application demonstrating how application components are implemented, and how they can be integrated
- Extension Points
  - Interfaces for creating families of functions that extend framework capabilities

Application Framework for Deployment



Application Framework Accomplishment

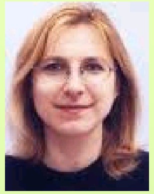


CHMM Format

Fixed Beam Width

CHMM GPU ObsProb

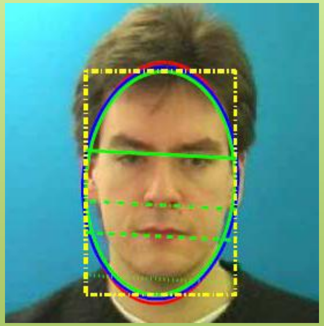
CHMM Scoring format



Prof. Dorothea Kolossa
Speech Application Domain Expert
Technische Universität Berlin

Extended *audio-only speech recognition* framework to enable *audio-visual speech recognition (lip reading)*

Achieved a ***20x speedup*** in application performance compared to a sequential version in C++



The application framework enabled a ***Matlab/Java programmer*** to ***effectively utilize highly parallel platform***

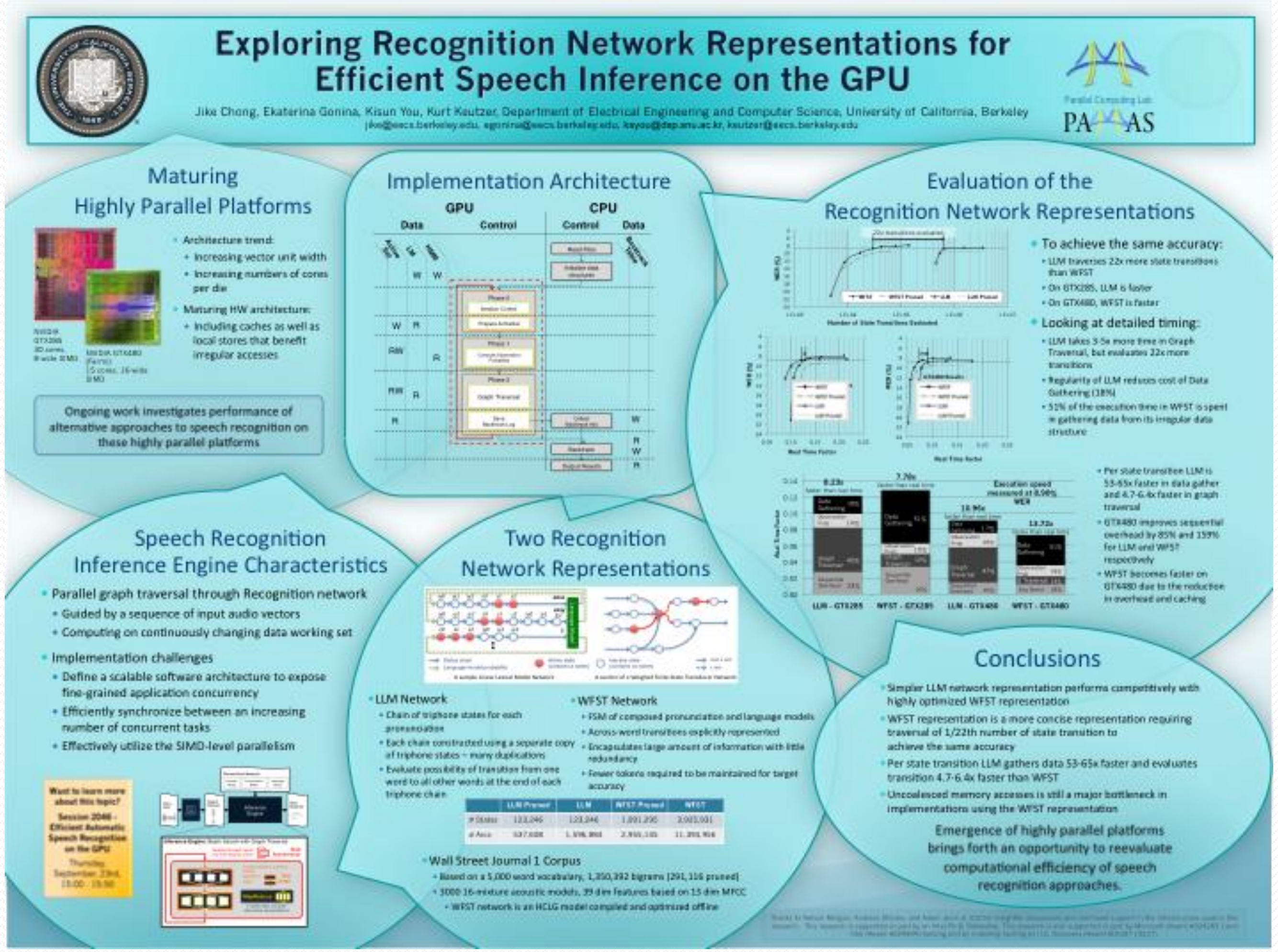
Dorothea Kolossa, Jike Chong, Steffen Zeiler, Kurt Keutzer, "Efficient Manycore CHMM Speech Recognition for Audiovisual and Multistream Data", Accepted at Interspeech 2010.

Key Lessons

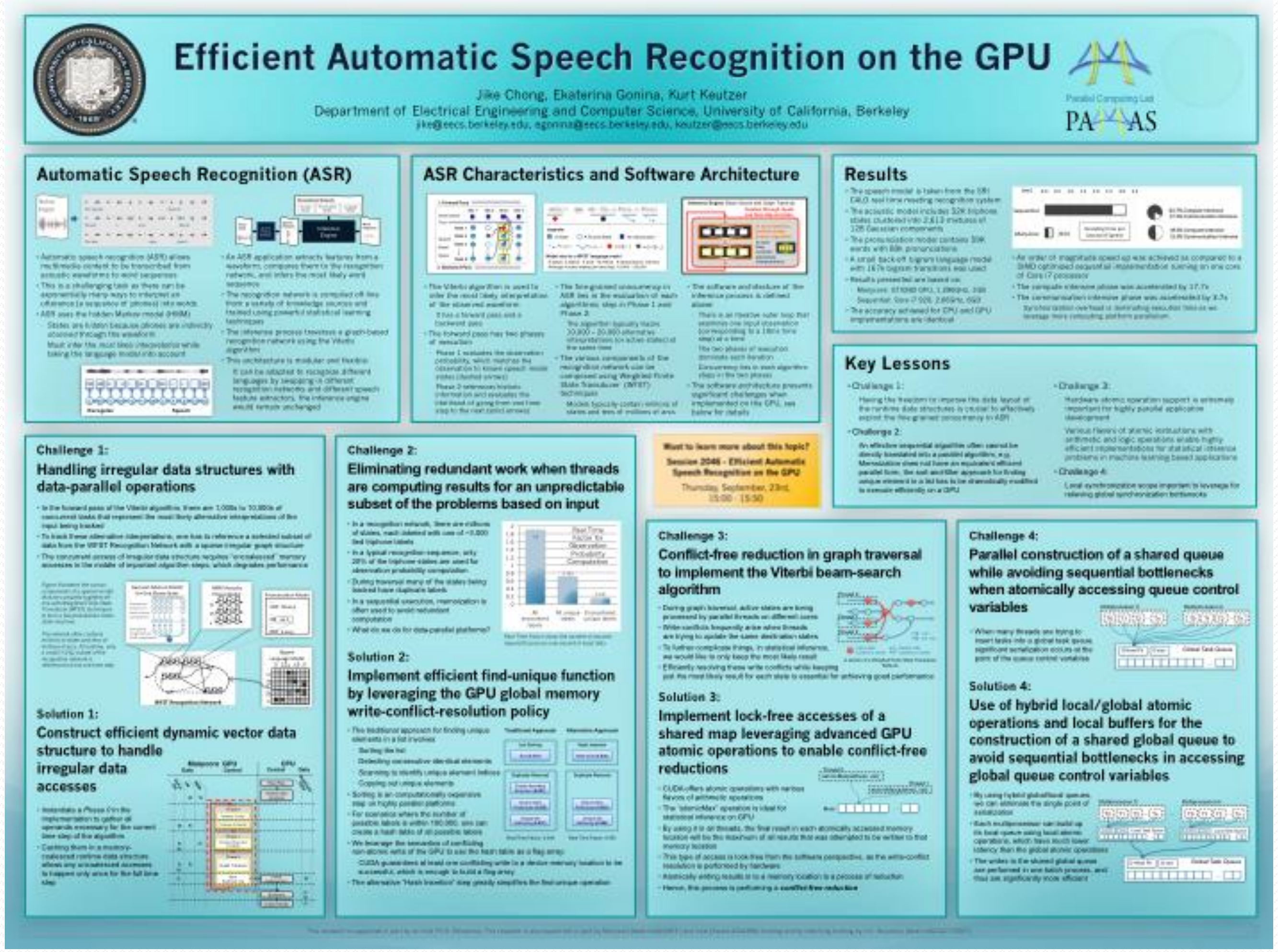
- Speech recognition application has many levels of concurrency
  - Amenable for order of magnitude acceleration on highly parallel platforms
- Fastest algorithm style differed for different HW platforms
  - Exploiting these levels of concurrency on HW platforms requires multiple areas of expertise
- Application frameworks help application domain experts effectively utilize highly parallel Microprocessors
  - Case study with an ASR application framework has enabled a Matlab/Java programmer to achieve 20x speedup in her application
  - Effective approach for transferring tacit knowledge about efficient, highly parallel software design for use by application domain experts
- Parallel computation is proliferating from servers to workstations to laptops and portable devices
  - increasing demand for adapting business and consumer applications to specific usage scenarios
- Application frameworks for parallel programming are expected to become an important force for incorporating hardware accelerators

Backup Slides

Audio Processing Poster: C01



Audio Processing Processing: C02



Programming Language & Techniques: R01