An FPGA Based Parallel Architecture for Music Melody Matching

Hao Wang and Jyh-Charn Liu

Computer Science and Engineering,
Texas A&M University

Outline

- Background
- Modeling of MIDI database
- Modeling of user query
- Approximate matching algorithm
- System architecture
- Evaluation

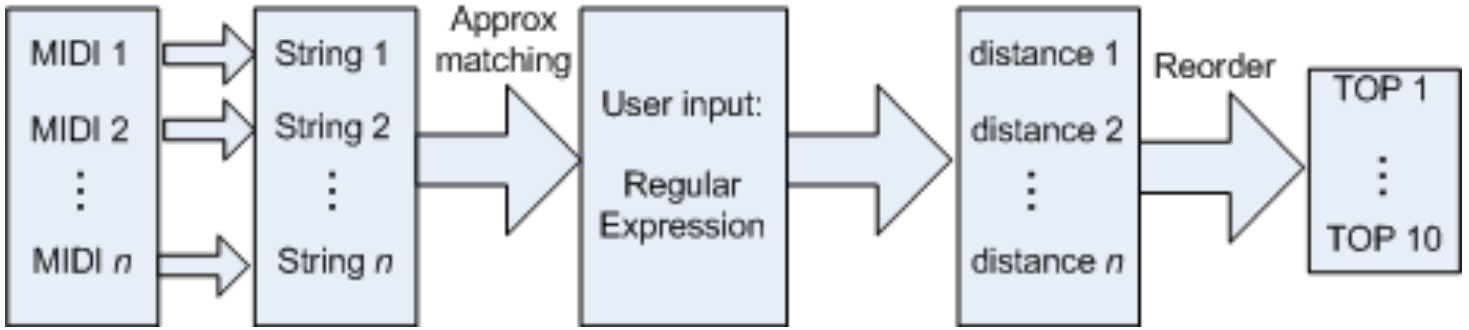
Background

- Explosive growth of music works
- Retrieval systems based on exact text metadata provide poor user experience
- Retrieval via acoustic signature (melody) is more natural and user friendly

Problem Definition

- The system aims to match the user humming/singing input against monophonic ground truth MIDI files, and find the most acoustically similar results.
  - User input is modeled as regular expressions (regexp)
  - MIDI files are modeled as strings

- The problem can then be modeled as approximate regular expression matching, where the similarity measure between a string and regexp is their edit distance.



Outline

- Background
- Modeling of MIDI database
- Modeling of user query
- Approximate matching algorithm
- System architecture
- Evaluation

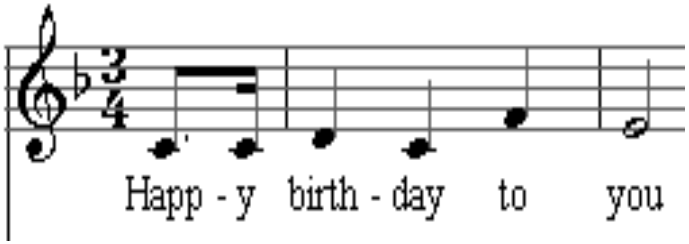
Modeling of MIDI File

- MIDI format specifies sheet music
- The software tool MIDICSV* can parse MIDI file into comma separated values.
- Using ASCII characters to denote pitch, and using 0.1 second as the unit time interval, we can model MIDI into string.

* http://www.fourmilab.ch/webtools/midicsv/

Modeling of MIDI File

Sheet music



CSV format

Time	Event	Value
0	Header	480
0	Tempo	600000
960	Note_on	60
1320	Note_off	
1320	Note_on	60
1440	Note_off	
1440	Note_on	62
1920	Note_off	
1920	Note_on	60
2400	Note_off	
2400	Note_on	65
2880	Note_off	
2880	Note_on	64
3840	Note_off	

ASCII symbols for pitch values

<<<<<<<>>>>>>><<<<<<AAAAAA@@
@@@@@@@@@@@@

Outline

- Background
- Modeling of MIDI database
- Modeling of user query
- Approximate matching algorithm
- System architecture
- Evaluation

Modeling of User Query

- Original user input .WAV file
- Estimate the fundamental frequency f_0 using YIN* algorithm
- Convert to MIDI pitches by

$$p = 69 + 12 \times log_2\left(\frac{f_0}{440\ Hz}\right)$$

*http://audition.ens.fr/adc/sw/yin.zip

Modeling of User Query

- Using ASCII characters to denote pitch, and use 0.1 second as the unit time interval, we can model user query into string.
- To ease later discussion of tempo variation, we group consecutive pitches that share the same value into a note.

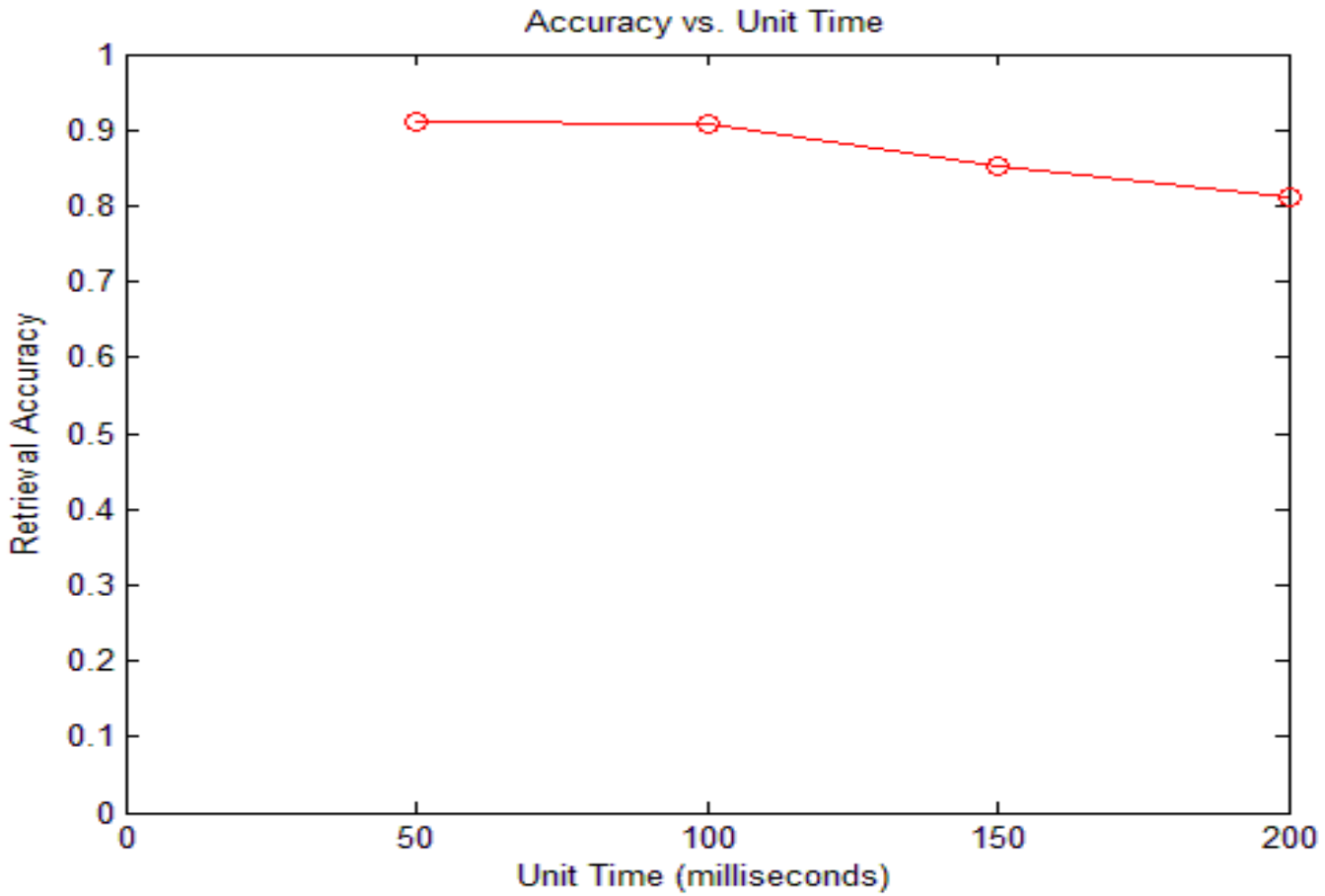
Modeling of User Query

- Key transposition support is a must
  - Adjust user query to the average pitch of MIDI being compared
  - Augment user query "key" up & down by 4: a total of 9 queries to represent a user input
- Duration (tempo) variation
  - Make the note's duration elastic: ranging from a half to twice of its original duration, by using the *constraint repetition* feature of regular expressions
  - Any match in the duration is a match

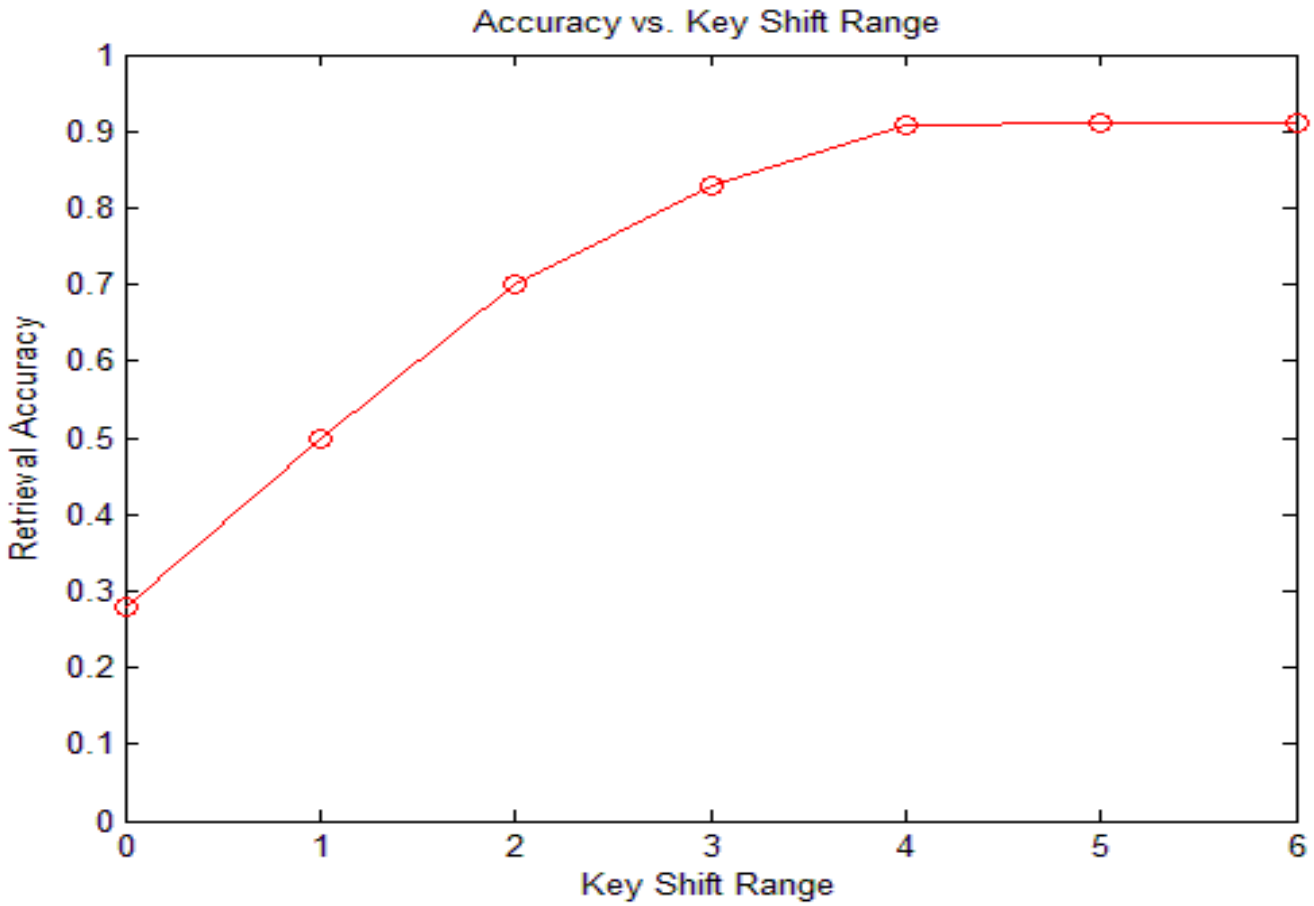
From User Humming to Regexp

- An original query string from YIN-MIDICVS:
  <<<<<<>>>>>><<<<<<AAAAAA@@@
  @@@@@@@@@@@@
- Note clustering
  <{6} • >{6} • <{6} • A{6} • @{12}
- Tempo-relaxed
  <{3,12} • >{3,12} • <{3,12} • A{3,12} • @{6,24}
- Denoted as CCR_1 • CCR_2 • CCR_3 • CCR_4 • CCR_5

Parameter Tuning: unit time



Parameter tuning: key shift range



Outline

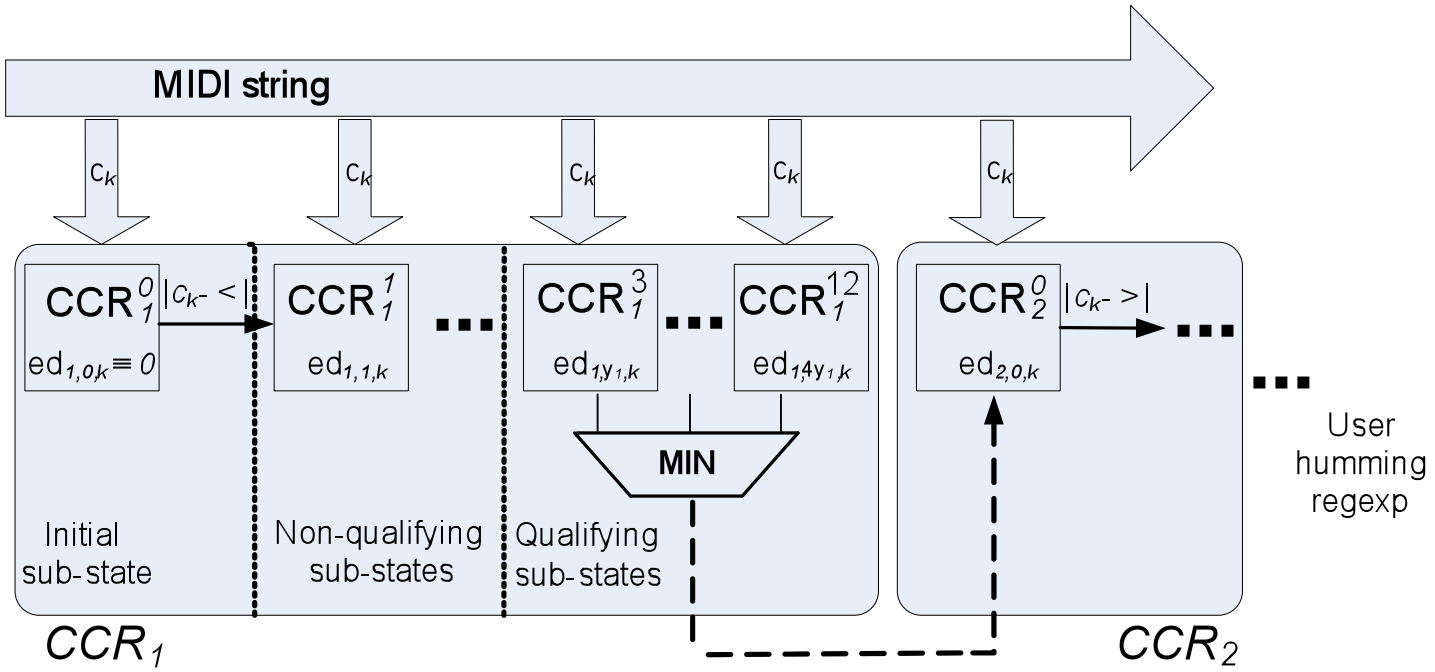
- Background
- Modeling of MIDI database
- Modeling of user query
- Approximate matching algorithm
- System architecture
- Evaluation

Approximate Matching Algorithm

- Classical dynamic programming algorithm based on a modified EDIT distance
  - Only consider substitution
  - No insertion/deletion:
    - Implicitly embedded into constraint repetition
    - They also introduce inter-state dependency during updates

State Transition Diagram

- CCR_1 : <{3,12} CCR_2 : >{3,12}



- All CCR sub-states match every input symbol independently.
- The MIN function calculates the local minimal matching of a CCR
- Substitution edit can be implemented by non-blocking assignment

Unitization of Constraint Repetitions

- Different constraint repetitions of CCRs lead to different execution path and processing time of MIN function
  - Make synchronization/parallelization inefficient
- Note that the upper bound of the constraint repetition is always 4 times the lower bound
  - Use {1,4} as an atomic constraint repetition to re-represent notes

Unitization of Constraint Repetitions

<{3,12} • >{3,12} • <{3,12} • A{3,12} • @{6,24}

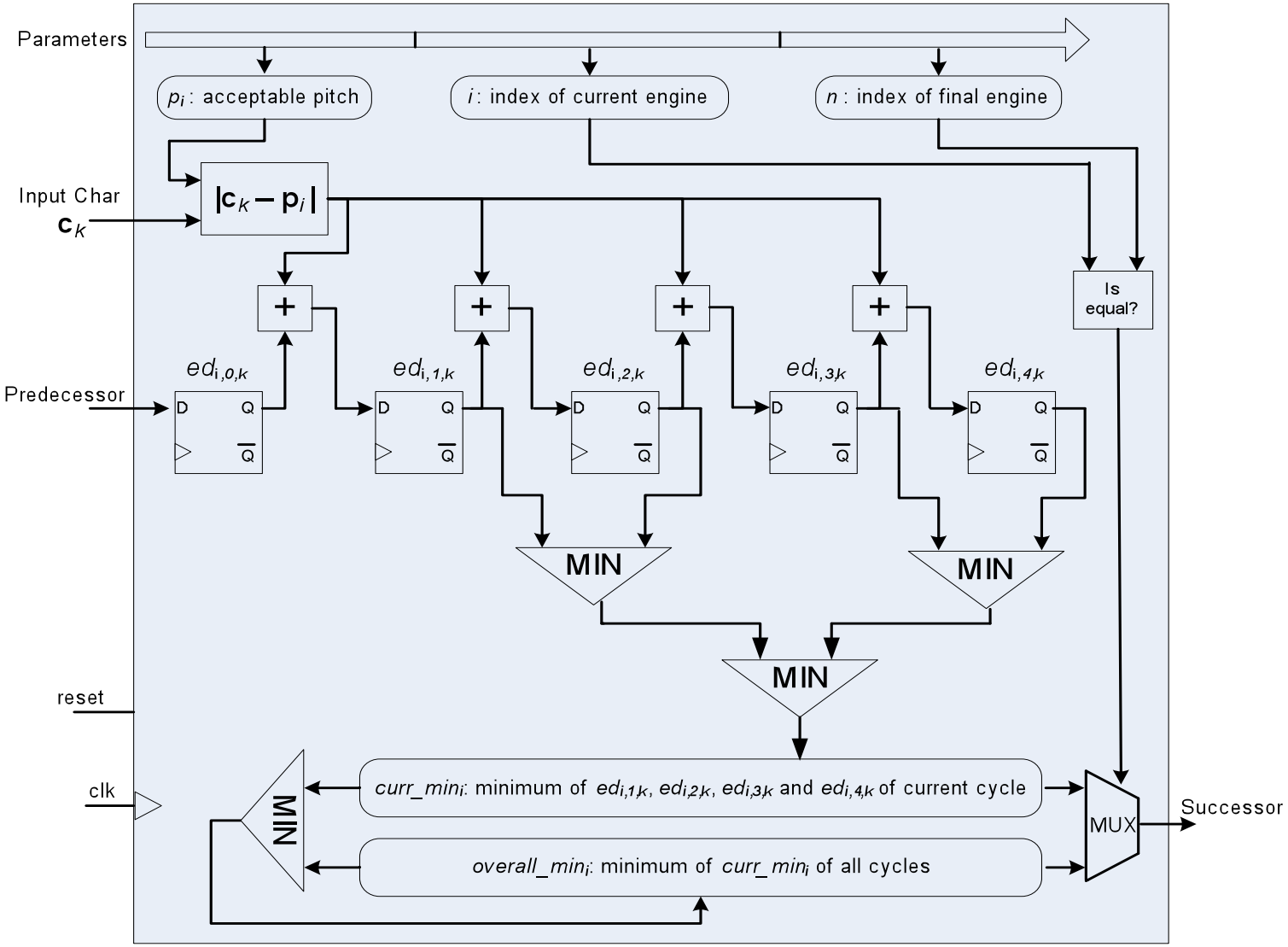
<{1,4} • <{1,4} • <{1,4} • >{1,4} • >{1,4} • >{1,4} • <{1,4} • <{1,4} • <{1,4} • A{1,4} • A{1,4} • A{1,4} • @{1, 4} • @{1, 4} • @{1, 4} • @{1, 4} • @{1, 4} • @{1, 4}

A-CCR

Outline

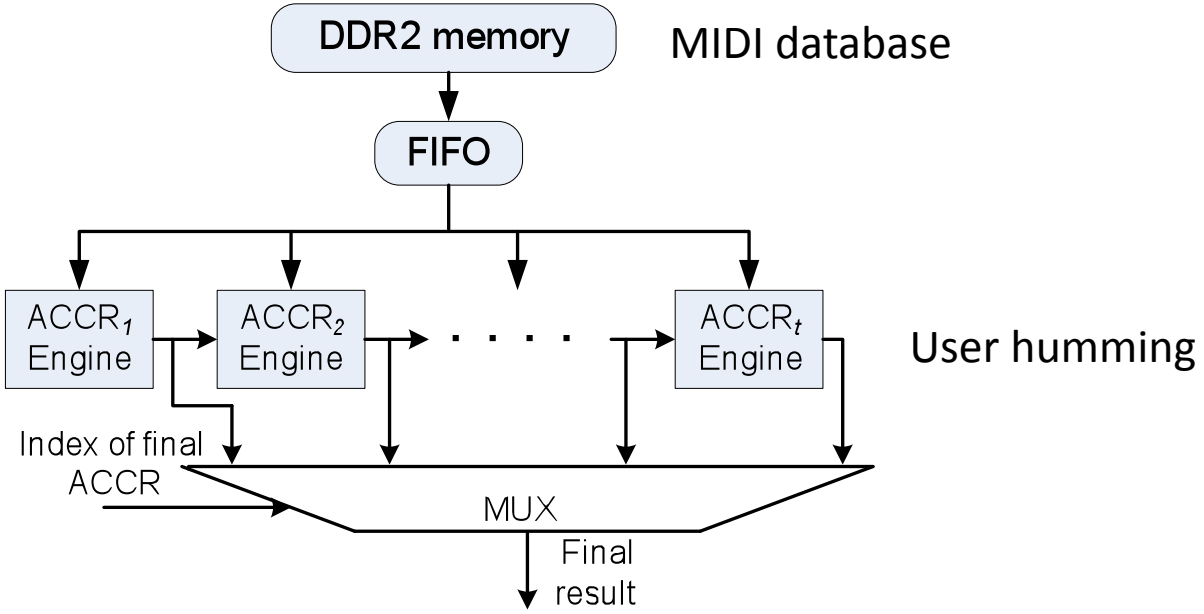
- Background
- Modeling of MIDI database
- Modeling of user query
- Approximate matching algorithm
- System architecture
- Evaluation

Atomic CCR (A-CCR) Engine



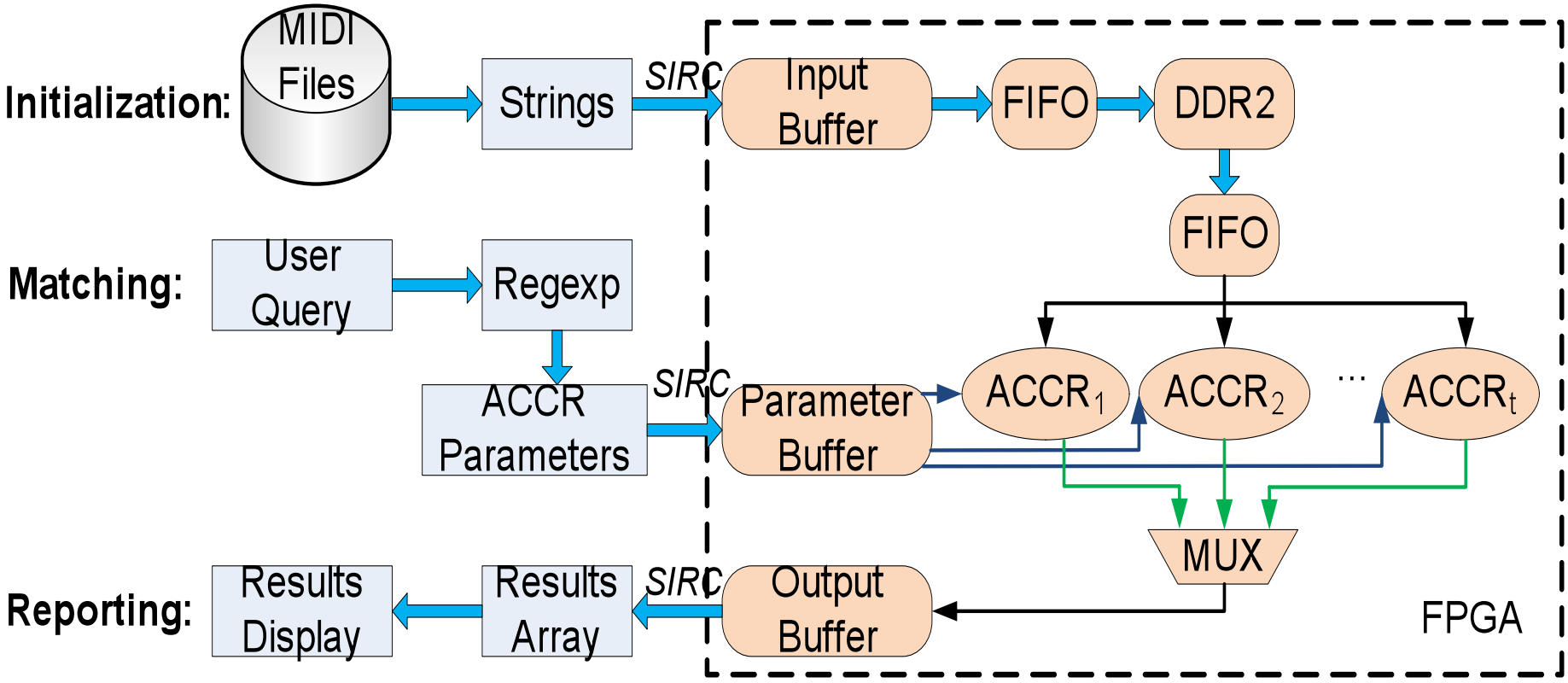
Melody Matching Engine

- Concatenated together, an array of ACCR modules implement a melody matching engine



- FIFO addresses cross-clock-domain issues
- Modular architecture, self-contained, highly scalable

System Integration and Flow Chart



SIRC: http://research.microsoft.com/apps/pubs/default.aspx?id=121293/

Outline

- Background
- Modeling of MIDI database
- Modeling of user query
- Approximate matching algorithm
- System architecture
- Evaluation

Evaluation: some facts

- The database has 5,569 MIDI files of Chinese pop songs and European folk songs. The generated text MIDI database is 1.62 MB in size.
- The user input usually has less than 100 ACCRs.
  - 9 variants are generated to handle key transposition
- The XUPV5 Virtex 5 FPGA can hold 300 ACCR engines at the utilization ratio of 87%.
  - It can process 3 variants of the user input at a time, takes 3 rounds to process all variants.
- The melody matching engine runs at 100 Mhz.
  - Long combinational logic paths (in MIN function)
  - Large MUX (100 8-bit inputs)
  - Long data path (FIFO – ACCR – MUX – Output BRAM)

Evaluation: runtime

- 355 user humming queries (IOACAS data set), starting from anywhere in the song, not necessarily from the beginning.
- Theoretical runtime
  - 1.62MB * 9 / 3 / 100MHz = 48.6 ms/query

    Database size * 9 variants/3-variants per round, at clock rate of 100Mhz
- The measured runtime,

	355 queries	a single query (avg.)
runtime	19.4 seconds	54.6 milliseconds

reflects the additional time for parameter downloading and results reporting

One time implementation cost

Implementation Step	Run Time
Synthesis	19 minutes
Map	2 hours and 15 minutes
Place & Route	6 hours and 19 minutes

Comparison

Method	Platform	Runtime	Accuracy	Speedup
CSJ2 [1]	Dual Intel Xeon Quad Core @ 2.0 GHz with 24 GB memory	210 min	86%	649X
HAFR1 [2]	Dual AMD Opteron Quad Core @ 2.0 GHz with 32 GB memory	247 min	80.6%	764X
YF2 [2]	Intel Core 2 Quad Core @ 2.40GHz with 8GB memory	367 min	90.4%	1135X
MME	XUPV5 FPGA @ 100MHz with 256 MB memory	19.4 sec	90.7%	--

http://www.midomi.com/, a commercial query-by-humming engine. Algorithm unpublished, platform unknown, matching accuracy and runtime comparable to MME.

[1] http://www.music-ir.org/mirex/wiki/2009:Query-by-Singing/Humming_Results
[2] http://www.music-ir.org/mirex/wiki/2010:Query-by-Singing/Humming_Results

Summary

- Key factors to achieve excellent combination of matching performance, power, and speed
  - CCR based modeling of the underlying structures of the problem
  - Parallel algorithm
  - FPGA technology
  - Parameterization and unification of ACCR engines
- We are expanding the CCR model to explore the relationship between CCR, DTW, and HMM

Thank you!