Fast Song Searching by Simultaneous Execution of HiFP2.0 and Staged LSH

Masahiro Fukuda
School of Information Science
Japan Advanced Institute of Science and Technology
Nomi, Ishikawa 923-1211
e-mail : fukuda-masahiro@jaist.ac.jp

Yasushi Inoguchi
Research Center for Advanced Computing Infrastructure
Japan Advanced Institute of Science and Technology
Nomi, Ishikawa 923-1211
e-mail : inoguchi@jaist.ac.jp

Abstract – Fingerprinting techniques are generally used to search a song quickly. In this paper, the fingerprint generation method HiFP2.0 and the identification method Staged LSH are combined and executed almost simultaneously. This method reduced around 3600 clock cycles and was about 8.54 % faster than the sequential execution of them in the case that the song of the query was a bit distorted by the lossy compression of MP3.

I. Introduction

Recent Internet provides an active field to distribute and enjoy music, while it is also being a hotbed of illegal copies. It is attractive to share songs on the Internet, but people feel uneasy because they must be always careful of laws and techniques for copyright protection.

The purpose of our research is to let such people share songs simply and legitimately. More specifically, as shown in Fig. 1, the routers by any Internet Service Provider (ISP) monitor packets and automatically detect registered songs, and then transmit the corresponding license and fee information to the receiver of the song file. This possibly allows the receiver to automatically pay the fee and obey the law. Enabling the router to search songs is the first step to realize the easy and legitimate way to share songs. In this system, only the users have to do is transmitting and receiving song files as plaintext through the router. AFP stands for Audio Fingerprint and it is described later.

The main difficulty in this system is the latency to search songs. The routers by ISPs must search songs from the large database quickly in this application. The speed of recent routers is 1, 10 or 40 Gbps, and even 400 Gbps is being developed. Then, for example, a song file of 1MB can pass 40 Gbps router in 0.18 ms if Song Data Extraction takes 10 % of the total latency and all other overheads are ignored. In other words, routers must complete searching in 0.18 ms. For achieving this, fingerprinting techniques are inevitable.

This paper introduces the audio fingerprint generation method named HiFP2.0 and identification method named Staged LSH, and then shows that they can be executed almost simultaneously to reduce total execution time at the cost of the small additional resource utilization.

The remaining of this paper is organized as follows. Section II introduces general explanations of fingerprinting technique and the hardware-oriented AFP generation method named HiFP2.0. Section III describes Staged LSH to quickly search AFP and our proposed method to execute HiFP2.0 and Staged LSH simultaneously. Section IV is the experiments and the resource utilization. Section V concludes the paper.

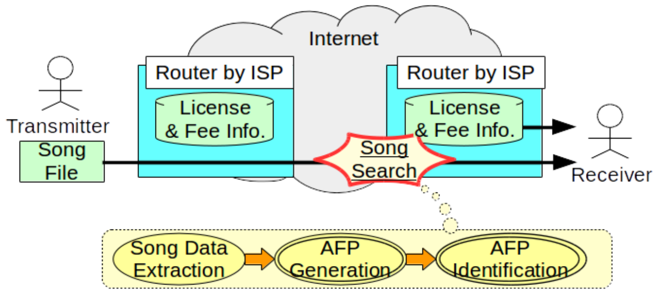


Fig. 1. Quick and legitimate song sharing system

II. Audio Fingerprint and Related Works

A. Audio Fingerprint

Audio fingerprint (AFP) is compact data which summarizes a song based on acoustic features. When you have a song file without any metadata and need to find it from the database about the vast number of songs, you should compare not the raw data of the song but the AFP of it.

The comparison process of AFP is faster than that of the raw data of songs because the size of AFP is smaller. The number of comparison instructions executed by processors becomes small and fast storage for the database might be available. Moreover, you do not need to prepare for many file types of songs such as WAV, MP3 or so on, because AFP techniques take advantage of acoustic features and are based on PCM data.

B. Related Works

So far, several AFP generation methods based on FFT or DWT have been proposed. The first paper on AFP as far as we know is [1]. It is based on Fourier transformation, but so it needs operations of floating-point numbers. Hence it is not so suitable for implementing on FPGA, compared with operations of integers.

Haitsma's method was improved by reducing the size of AFP [2]. This was done by finding the location where the probability of distortion is low in AFP. However, it takes advantage of Haitsma's method and is not the best choice to accelerate the speed of AFP generation by FPGA.

Martinez et al. implemented audio fingerprinting technique on FPGA and GPGPU [3]. But this still calculates FFT, and the exact latency was not mentioned in the paper. There are very limited research papers on the AFP technique which is accelerated by FPGA or the other hardware-specific methods.

HiFP2.0 is another method to generate AFP from a song, which was developed by Araki [4]. Fig. 2 is the outline of the process of HiFP2.0. It generates 4096-bit AFP from PCM data of a song based on Discrete Wavelet Transformation (DWT) by Haar.

HiFP2.0 is hardware-oriented because it takes advantage of Haar Wavelet Transformation and only needs addition, shift operation and comparison of integers. Fig. 3 and Fig. 4 are the detailed algorithms of MHWT in Fig.2 and HiFP2.0 respectively, which are quoted and a bit revised from [4].

In our implementation, the number of clock cycles needed for HiFP2.0 is 4109 by pipelining it and generating 1 bit of AFP per 1 clock cycle, where PCM data were cyclically divided into a few BRAMs and were read in parallel. Further parallelization of HiFP2.0 is possible, but we did not apply it this time in order to save resources of FPGA and to simplify the explanations.

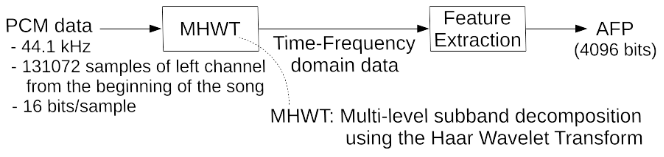

PCM data
- 44.1 kHz
- 131072 samples of left channel from the beginning of the song
- 16 bits/sample
MHWT
Time-Frequency domain data
Feature Extraction
AFP (4096 bits)
MHWT: Multi-level subband decomposition using the Haar Wavelet Transform

Fig. 2. Outline of HiFP2.0 process

```
MHWT(wav[] ← Input signal,
       n ← Number of samples of input signal,
       m ← Number of output samples) {
  for (; n > m; n /= 2) do
    for (i = 0; i < n/2; i++) do
      Hi[i] = (wav[2*i] - wav[2*i+1])/2;
      Lo[i] = (wav[2*i] + wav[2*i+1])/2;
    end for
    wav[] ← Lo[];
  end for
  return (Hi, Lo);}
```

Fig. 3. The algorithm of Multi-level subband decomposition using the Haar Wavelet Transformation (MHWT)

```
HiFP2.0(wav[] ← PCM data) {
  n ← 131,702; /* Number of samples of input signal */
  m ← 16,384; /* Number of output samples */
  Hi[], Lo[] ← MHWT(wav[], n, m);
  j ← 0;
  for (i = 0; i < m - 4; i += 4) do
    tmp = Lo[i] - Lo[i+4];
    if tmp > 0 then
      AFP[j] = 1;
    else
      AFP[j] = 0;
    end if
    j++;
  end for
  AFP[m/4-1] = 0;
  return AFP;}
```

Fig. 4. The algorithm of HiFP2.0

III. Proposed Method: Staged LSH over HiFP2.0

A. Staged LSH

Staged LSH is a method to quickly identify AFP which is the most similar to that of the transmitting song file (the query) through the router, which was developed by Yang [5].

This takes advantage of hash search, and the execution time of hash search is theoretically constant when there is not any collision of the hash values or any bit error in AFP of the query.

Fig. 5 shows 3 types of data unit which are used in Staged LSH. One is AFP and the remaining two are Sub-Fingerprint (Sub-FP) and Frame. Sub-FP is a 32-bit compartment of an AFP and Frame is a sequential triplet of Sub-FPs.

Fig. 6 shows a general process of Staged LSH. It repeats 3 steps named Hash Search, Coarse Search and Exact Search 126 times at most. Hash Search and Coarse Search in each loop refers fidx-th frame: Frame[fidx], where fidx is the loop number (0 for 1st loop and 125 for final loop).

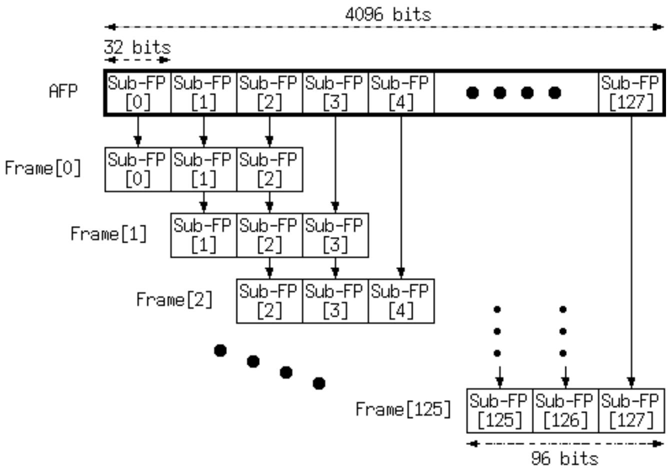


Fig. 5. Relationship of AFP, Sub-FP and Frame

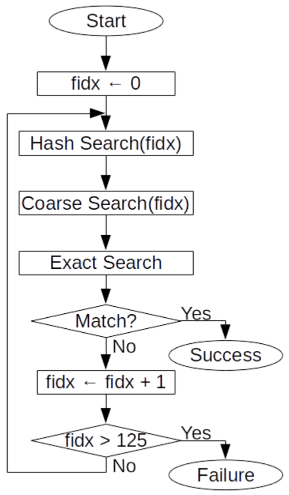


Fig. 6. Outline of Staged LSH process

Hash Search calculates a hash value from a frame of the query and finds all frames in the database of AFPs (FPDB) whose hash value is same.

Coarse Search checks hamming distance between the frame of the query and those of AFPs in FPDB. The only AFPs in FPDB whose hamming distance is equal to or less than a threshold can pass Coarse Search.

Exact Search also checks hamming distance, but the range is not a frame but whole AFP. It compares not 96 bits but 4096 bits and the threshold is different from Coarse Search.

The actual execution time of Staged LSH depends on the latency of memory accesses. Staged LSH might require large memories for the hash table and FPDB. These data might be more than 10 gigabytes in the case that it supports 3500 songs like Google Play Music or Apple Music. The latency to access such large memories usually varies and is not small.

Our implementation uses PCI Express (PCIe) 3.0 x8. The theoretical speed of PCIe of the FPGA board which we used, named VC709 connectivity kit, is 5.72 GB/s [6]. But in our current implementation, the measured speed was 26.7 MB/s and so the latency to read 1 DW (32 bits) was 0.15 us.

The latency of Staged LSH also depends on the distortion of the query. The PCM data of the query are sometimes distorted by MP3 compression. And this prevents Staged LSH from completing its process in the first loop, mainly because Hash Search fails. So it is also important in Staged LSH to evaluate the performance in the case of the distortion.

In our implementation of the hash table, Hash Search and Coarse Search generally need to read (2 + 1.02N) DWs from FPDB in total, where N is the number of entries in a bucket and an entry is the address of a frame whose hash value is same. And Exact Search needs to read 128M DWs where M is the number of AFP which passes Coarse Search. Therefore, the total execution time of Staged LSH is around (2 + 1.02N + 128M) * L * 0.15 us, where L is the number of loops to find AFP enough similar to that of the query.

B. *Overlap of Stage LSH and HiFP2.0*

Actually, HiFP2.0 and Staged LSH can be executed almost simultaneously. That seems impossible by a naive thought because HiFP2.0 outputs 4096-bit AFP or 128 Sub-FPs and Staged LSH requires full of them. But the step in Staged LSH which uses all 128 Sub-FPs is only Exact Search. Hash Search and Coarse Search need only 3 Sub-FPs in each loop.

Fig. 7 shows the brief interface between HiFP2.0 and Staged LSH in the proposed method. The both modules are executed almost simultaneously. HiFP2.0 generates a SubFP per 32 clock cycles and stores it to the Block RAM, and set a corresponding bit in SubFP_av flags at the same time to let Staged LSH module recognize the SubFP is available.

In Fig. 7, when a Frame becomes available, Staged LSH can execute Hash Search and Coarse Search. And if there is any Frame in FPDB which deserves to go to Exact Search, Staged LSH waits until all the bits in SubFP_av are set and then goes to Exact Search. If there was no Frame in FPDB similar to that of the query in Hash Search and Coarse Search, Staged LSH waits until the next Frame is available and then repeats Hash Search and Coarse Search.

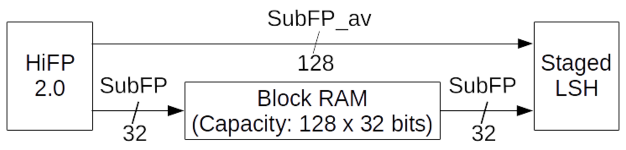


Fig. 7. Outline of the proposed method

The proposed method can be at most 2 times faster than the situation that Staged LSH starts after HiFP2.0 finishes. The best case is that the execution times of HiFP2.0 and Staged LSH are same.

This method rarely increases the delay of HiFP2.0 or Staged LSH because it just adds SubFP_av and sets/waits one of the flags each time 32 bits of AFP are generated.

The probability that Staged LSH outputs a wrong answer in the case of that the query is distorted because of MP3 compression, cannot be worsened by the proposed method, because this method maintains the data dependencies.

IV. Evaluation

A. *Experimental Conditions*

We carried out an experiment by real machines to evaluate the proposed method.

Table I is the basic experimental conditions. In this experiment, 10 million song files were generated by random numbers beforehand and each of them was 1MB and omits noise 6 seconds. 100 queries were randomly determined from the 10 million song files beforehand. As queries, distorted song files were prepared as well as original song files by lame 3.99.5 program. The commands for attaching the distortion are as follows:

```
lame -b 128 original.wav temporal.mp3
lame --decode temporal.mp3 distorted.wav
```

Table II is the parameters of Staged LSH. We constructed FPDB and the hash table, which are almost 10GB in total, based on original song files above, where the bit length of the hash values is 24. Provided that all frames can be assumed as random numbers, the number of entries in each bucket will be N = 126 x 10^7 / 2^{24} = 75.1.

The thresholds in Table II mean the upper limit of hamming distance to pass each step of Staged LSH. Provided that AFPs can be assumed as random numbers, the probabilities that a different song passes Coarse Search and Exact Search are 4.85 x 10^{-7} % and 4.33 x 10^{-235} % respectively. Because the probability about Coarse Search is so small, the number of AFP which passes it is also negligible except for the correct AFP. As for Exact Search, the probability that any different song from ten million songs can pass is 2.16 x 10^{-215} % even if the birthday paradox is considered.

Table III shows the device and the clock frequency to be used in the experiments. Table IV and V are about the development tool for FPGA. The board we used was VC709 Connectivity Kit by Xilinx Inc. and the parts in it were unmodified. The tool which supports VC709 is Vivado. PCI Express (PCIe) 3.0 x8 is available in VC709 and the FPGA can communicate with a desktop PC through the IP core. The clock frequency to control the IP core for PCIe is 250 MHz and we used this clock also for executing HiFP2.0 and Staged LSH. BRAM in Table V is used for Base Address Register 0 (BAR0) memory region of PCIe, by which the desktop PC lets VC709 know the beginning of the logical address of FPDB.

TABLE I
Basic Experimental Conditions

Parameter	Value
Content of Audio File	Random Number
Size of Query	1 MB
Number of AFPs in FPDB	10 Million
Number of Queries	100
Distortion	Encoding / Decoding MP3 128bps by lame 3.99.5

TABLE II
Parameters of Staged LSH

Parameter	Value
Bit Length of Hash	24
Threshold in Coarse Search	24
Threshold in Exact Search	1024

TABLE III
Execution Environment

Item	Description
Board	VC709 Connectivity Kit
Device	XC7VX690T-2FFG1761C
Clock Frequency	250 MHz

TABLE IV
Development Environment

Item	Description
Machine	ThinkPad X240
Operating System	Ubuntu 14.04.4 LTS
Kernel	3.16.0-71-generic (x86_64)
Design Tool	Vivado HLS 2016.1

TABLE V
Vivado IP Cores

Purpose	IP Core
PCI Express	Virtex-7 FPGA Gen3 Integrated Block for PCI Express 4.2
BRAM	Block Memory Generator 8.3 (Rev. 2)

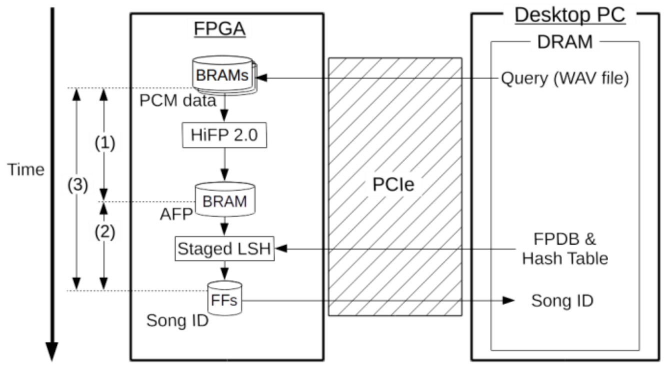
Vivado 2016.1 also has High Level Synthesis (HLS) function and we took advantage of it for implementation of HiFP2.0 and Staged LSH modules. To let them use PCIe, we applied manual interfaces in HLS to activate the modules which we made for controlling IP Core of PCIe in Verilog HDL, referring to an example design by Vivado. As for BRAM of SubFP, it is dual-port and written in Verilog HDL. One side of this BRAM is connected to HiFP2.0 module as RAM_1P_BRAM and the other side is connected to the Staged LSH module as ROM_1P_BRAM. SubFP_av is also specified as a manual interface. It is written by HiFP2.0 module and read by Staged LSH module.

As for the desktop PC, the motherboard is MW50-SV0 by GIGABYTE and its interfaces to memories (32GB) are DDR4. We made and used a device driver which probes VC709 with the specified vendor ID and device ID of PCIe, allocates consecutive and coherent memory regions, and lets the application software write FPDB and a query in the region and order VC709 to start searching a song and read the result.

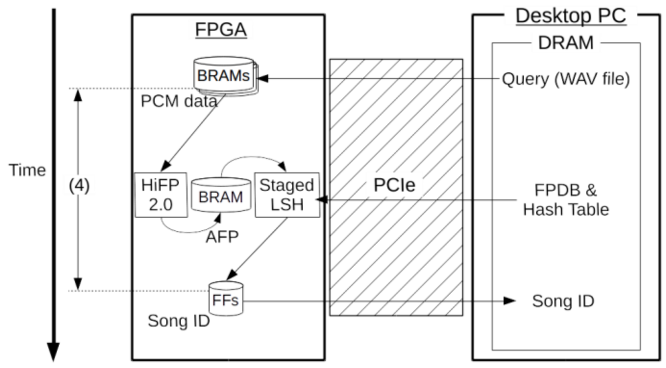
B. *Experimental Results*

We measured the average numbers of clock cycles to execute (1), (2), (3) and (4) shown in Fig. 8 in the conditions of Table I, II, III, IV and V. (1), (2) and (3) are about the situation that disables the proposed method, while (4) is about the proposed method. First, the desktop PC orders FPGA to start searching a song (by writing BAR0 memory) and then FPGA loads the query (WAV file of the song) to a BRAM, but this process is not measured. The first measured time is "(1) HiFP2.0." It is between the beginning of reading PCM data from a BRAM and the end of writing AFP to another BRAM. The next is "(2) Staged LSH." It is measured after (1) completed and is between the beginning of reading AFP from the BRAM and identifying the song. "(3) HiFP2.0 then Staged LSH" is just the addition of (1) and (2). "(4) Overlap of HiFP2.0 and Staged LSH" is the time between the beginning of reading PCM data from the BRAM and the end of identifying the song by the proposed method.

Table VI is the experimental results which show (1), (2), (3) and (4). The differences between (3) and (4) were 3341.01 clock cycles for original song files and 3645.24 clock cycles for distorted song files respectively.



(a) Naive method (sequential execution)



(b) proposed method

Fig. 8. Time range to measure

TABLE VI
Number of Clock Cycles for HiFP2.0 and Staged LSH
(The numbers between brackets mean the standard deviations)

Process	Average Number of Clock Cycles	
	Original	Distorted
(1) HiFP2.0	4109.00 (0)	4109.00 (0)
(2) Staged LSH	9506.08 (2687.36)	42215.18 (33207.42)
(3) HiFP2.0 then Staged LSH	13615.08 (2687.36)	46324.18 (33207.42)
(4) Overlap of HiFP2.0 and Staged LSH	10274.07 (2374.18)	42678.94 (33117.64)
(5) Difference = (3) – (4)	3341.01 (328.34)	3645.24 (617.18)
(6) Improvement ratio = (3) / (4) – 1	32.5 %	8.54 %

The number of failing to output the correct answer was zero in this experiment, even when the query was distorted. All 100 queries were correctly identified by both the sequential method and the proposed method.

In summary, the proposed method achieved to reduce around 3600 clock cycles, especially in the case that song files were distorted. 3600 clock cycles (14.4 us at 250 MHz) are almost same as the number of clock cycles needed by HiFP2.0. That means that HiFP2.0 and Staged LSH are successfully executed simultaneously. Further, 42678.94 clock cycles and 250 MHz in the case of distorted songs are corresponding to 0.171 ms and less than 0.18 ms for 40 Gbps described in the Introduction section.

C. Resource Utilization

Table VII is the resource utilizations of (3) and (4). The increase ratio of (4) to (3) was only 1.16 % on LUT and 1.79 % on FF. Those are small compared with the reduction of the latency (32.5 % or 8.54 %).

TABLE VII
Resource Utilizations

Element	(3) HiFP2.0 then Staged LSH	(4) Overlap of HiFP2.0 and Staged LSH	Ratio of increase
LUT	35268 (8.14 %)	35677 (8.24 %)	1.16 %
LUTRAM	56 (0.03 %)	56 (0.03 %)	0 %
FF	20977 (2.42 %)	21352 (2.46 %)	1.79 %
BRAM	52 (3.54 %)	52 (3.54 %)	0 %
IO	5 (0.59 %)	5 (0.59 %)	0 %
GT	8 (22.22 %)	8 (22.22 %)	0 %
BUFG	5 (15.63 %)	5 (15.63 %)	0 %
MMCM	1 (5.00 %)	1 (5.00 %)	0 %
PCIe	1 (33.33 %)	1 (33.33 %)	0 %

V. Conclusions and Future Works

To achieve a song sharing system by the routers by ISPs, we proposed a method to execute HiFP2.0 and Staged LSH simultaneously, which generates and identifies audio fingerprints respectively. The experimental result shows that the total number of clock cycles is about 3600 less than the situation that Staged LSH starts after HiFP2.0 completely finishes, especially in the case that the query is distorted. The improvement was about 8.54 % and this can help the song sharing system speedup to 40 Gbps.

One of the future works is out-of-order execution of HiFP2.0 and Staged LSH. Packets do not always arrive in the correct order in Internet Protocol (IP), so executing HiFP2.0 and Staged LSH little by little per arrival of a packet can lead the improvement of performance. Another future work is to carry out the experiment with song files more similar to the real songs. In this paper, we just used random numbers and the generated songs were almost the noise. To make our research more practical, experimental data should have some melodies, harmonies and rhythms at least.

References

[1] Jaap Haitsma and Ton Kalker, “A Highly Robust Audio Fingerprinting System,” *Proceedings of the International Society for Music Information Retrieval (ISMIR)*, pp. 107-115, 2002.

[2] Hendrik Schreiber and Meinard Muller, “Accelerating Index-Based Audio Identification,” *IEEE Transactions on Multimedia*, Volume 16, Issue 6, 2014.

[3] Jose Ignacio Martinez, Jaime Vitola, Adriana Sanabria, Cesar Pedraza, "Fast Parallel Audio Fingerprinting Implementation in Reconfigurable Hardware and GPUs," *Proceedings of the Southern Conference on Programmable Logic (SPL)*, pp. 245-250, Apr. 2011.

[4] Koichi Araki, Yukinori Sato, Vijay Jain and Yasushi Inoguchi, "Performance evaluation of audio fingerprint generation using Haar wavelet transform," *Proceedings of the 2011 International Workshop on Nonlinear Circuits, Communications and Signal Processing (NCSP11)*, pp. 380-383, 2011.

[5] Fan Yang, Yukinori Sato, Yiyu Tan and Yasushi Inoguchi, "Searching acceleration for audio fingerprinting system," *Proceedings of the 2012 Joint Conference of Hokuriku Chapters of Electrical Societies (JHES)*, F-15, 2012.

[6] Xilinx, "Virtex-7 FPGA XT connectivity targeted reference design for the VC709 board (UG962 (v3.0))," 2014.