Probabilistic Strategies Based on Staged LSH for Speedup of Audio Fingerprint Searching with Ten Million Scale Database

Masahiro Fukuda[1,3], Yasushi Inoguchi[2]

[1] School of Information Science, Japan Advanced Institute of Science and Technology

[2] Research Center for Advanced Computing Infrastructure, Japan Advanced Institute of Science and Technology

[3] Electronics and Information Engineering, National Institute of Technology Ishikawa College

ABSTRACT

We are developing and improving algorithms to identify audio fingerprints (AFP) in a network router. Staged Locality Sensitive Hashing (LSH) is one of them and nearly as fast as 1Gbps of prevalent network routers. In this paper, we propose two extensions from Staged LSH, both of which take advantage of probabilistic strategies. One is Neighbor Staged LSH, which is to tune up about how to choose buckets for the hash method for searching. The other is Hierarchy Staged LSH, whose strategy is to focus on the popularity of songs. Adopting both achieved at most 182.8 times as fast as the simple Staged LSH and it was equivalent to 1 Gbps. The accuracy rate was 100 % if the BER of AFP is less than 15 %.

1. INTRODUCTION

It is known that there are at least 10 million songs on the Internet. For example, Apple Music has over 30 million songs [1]. To protect songs against illegal copies, Digital Rights Management (DRM) and so on are being used. Other techniques like watermarking are also famous.

However, these techniques tend to lead to inconvenience. DRM restricts the playing environment and watermarking deteriorates the acoustic information. They are harmful for availability, while they are necessary to protect copyrights.

Figure 1 is the system which we are developing. It can quickly identify the song passing in a network router to automatically send the license and payment information. It aims to be a key component technique to improve convenience for legitimate distribution of songs to non-malicious users, setting aside about protecting copyrights for the time being.

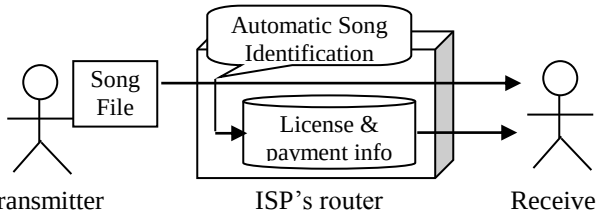

Figure 1. Auto song identification in network router

Figure 2 shows the tasks of ISP's router on the system. The system takes advantage of the audio fingerprint (AFP) technique to identify songs faster. Nevertheless, the most difficult part in the tasks is searching AFP, because the database of 10 million songs requires a slow storage.

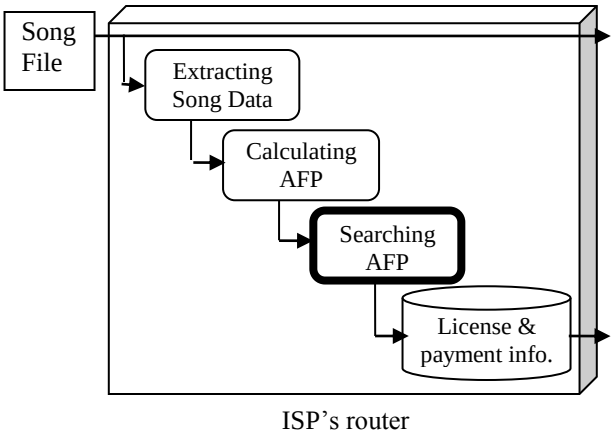

Figure 2. Basic function of HiFP2.0

The system is assumed to be embedded in a network router, so the searching speed should be no less than 1 Gbps, or the most prevalent in the standard home. More specifically, if a song file is 1 MB, the speed of network routers is 1 Gbps and the clock frequency of the processor is 250 MHz as typical parameters, then the system should complete to identify a song within 8 milliseconds or 2 million clock cycles without consideration of overheads such as packet headers.

The main purpose of our research is the speedup of identifying the song from the 10 million scale database to make the practical system. Especially in this paper, we focus on the searching audio fingerprints (AFPs), which is more technically difficult part compared with extracting audio data from traffic data and calculating AFPs from audio data.

The paper is organized as follows. Section 2 reviews the related works on AFP. It especially focuses on the hardware accelerated AFP calculation method named HiFP2.0. Section 3 reviews the previous research named Staged LSH. Section 4 shows the proposed methods, named NLSH and

HLSH, and their implementations in this paper. Both are based on probabilistic strategies. Section 5 describes about the experiments and their results. Finally, Section 6 concludes this paper.

2. RELATED WORKS ON AFP

2.1 Audio Fingerprint

2.1.1 General AFP

An audio fingerprint (AFP) is a key technique to quickly identify a song. AFP is a compact expression to summarize acoustic information of a song. It reduces comparison operations, improves fast memory efficiency and omits to prepare multiple file formats for songs.

The research on AFP can date back to Haitsma and Kalker's [4] as far as we know. It was not still practical due to its huge size of the database of AFPs. Hendrik and Meinard proposed the method named SysEL and SysSS [6] and they reduced the size by extracting only important features to identify songs. But the best method (SysSS) is still estimated to require 22 gigabytes for 10 million songs.

2.1.2 HiFP2.0

Currently, we are adopting HiFP2.0 as the AFP calculation method. It is an algorithm to calculate a 4096-bit AFP from only the beginning 2.97-second of a song. That means that the size of the database is 5 gigabytes for 10 million songs.

Figure 3 shows the outlined function of HiFP2.0. The input is Pulse Code Modulation (PCM) data in the beginning 2.97 seconds of a song and the output is a 4096-bit AFP. The PCM data must be stereo, 44.1 kHz sampling frequency and 16-bit quantization bit length.

Figure 3. Basic function of HiFP2.0

The most significant feature of HiFP2.0 is being hardware-oriented. HiFP2.0 is based on Haar Wavelet Transform and only needs addition, shifting and comparison operations. It also can be highly parallelized. We should take advantage of HiFP2.0 and also execute searching AFPs on the same FPGA to make them a hardware accelerator.

2.2 Fingerprint Searching

2.2.1 LSH

Locality Sensitive Hashing (LSH) is a method to calculate a hash, whose number of dimensions are decreased by a probabilistic process [3]. A kind of LSH is extracting several bits from the input and just gathering them. Staged LSH described later also adopts this simple calculation and compares the similarity of hashes by the hamming distance.

2.2.2 Other LSH

As another kind of LSH methods, Batch-Orthogonal Local-Sensitive Hashing (BOLSH) was applied to multimedia by Gao et al [5]. It provides accurate estimation by dividing the input space into regular regions and reducing the variance. By applying BOLSH, Gao et al achieved higher performance and an accuracy rate to recognize movies. However, for our purpose, such an advanced method is not necessary because the data structure of AFPs, especially calculated by HiFP2.0, is not complex. To reduce the resource utilization, we are applying another, more simple method.

2.2.3 Staged LSH

Staged LSH is an algorithm developed by Yang [8] to find the AFP whose hamming distance is the closest to the AFP of the query. The input is a query AFP and the output is the AFP from the database which is the closest to the input.

2.2.4 Algorithm of Staged LSH

Figure 4 is the algorithm of Staged LSH. It has three steps to gradually decreasing the search scope.

1. Locality Sensitive Hashing (LSH)
2. Coarse Search (CS)
3. Exact Search (ES)

```
1  for a frame frame_q in a query AFP_q do
2    hash <= LSH(frame_q)                                  } LSH
3    bucket <= get_bucket(HashTable, hash)                 }
4    for a frame index fidx in bucket do                   }
5      BestMatch.hd = 4096                                 }
6      frame_d <= get_frame(Database, fidx)                } CS
7      hd <= hamming_distance(frame_q, frame_d)            }
8      if hd ≤ Th1 then                                    }
9        AFP_d <= get_AFP(Database, fidx)                  }
10       hd <= hamming_distance(AFP_q, AFP_d)              }
11       if hd ≤ Th2 and hd ≤ BestMatch.hd then            }
12         BestMatch.hd <= hd                              }
13         BestMatch.AFP <= AFP_d                          } ES
14       endif                                             }
15     endif                                               }
16     if BestMatch.hd ≠ 4096 then                         }
17       return BestMatch.AFPx                             }
18     endif                                               }
19   done
20 done
21 return None
```

Figure 4. Algorithm of Staged LSH

LSH just calculates the hash of the first frame of the query. The hash specifies the set of frames, named the **bucket,** in the database to be used by CS. Yang adopted 13 as the bit length of the hash.

CS examines that the hamming distances of the frames specified by LSH are enough close to the first frame in the query. The threshold of the hamming distance is named Th_1.

ES examines that the hamming distance of the AFPs whose frames pass CS are enough close and the closest to the 4096-bit query. The threshold is named Th_2.

If there are bit errors in the query and no AFP in the database passed ES, then Staged LSH tries LSH, CS and ES again with the second frame of the query. It will repeat until the final frame unless any AFP passes ES.

2.2.5 Problem of Staged LSH

Yang's research [8] had two problems.

1. Only 300 AFPs were used in the experiments.
2. Staged LSH is slowed down by bit errors in the query.

3. EXTENSIONS OF STAGED LSH

In this paper, we propose two methods to tackle the problems of Staged LSH. One method is Neighbor Staged LSH (NSLSH) and the other is Hierarchy Staged LSH (HSLSH). The both methods are based on Staged LSH, but takes advantage of the probabilistic biases.

3.1 Neighbor Staged LSH

NSLSH is an extension about how to choose buckets. The simple Staged LSH chooses only a bucket with 13-bit hash and then proceeds to the CS step. NSLSH chooses 25 buckets with 24-bit hash, whose frame's hashes are same or 1-bit different.

Figure 5 is the comparison between the simple Staged LSH and NSLSH. Underlines show the differences from the hash which was calculated by LSH.

Simple Staged LSH in [8]

Hash | 0001001101011 |

⇨ Choose the bucket of 0001001101011

Neighbor Staged LSH

Hash | 010101100000101001110011 |

⇨ Choose the buckets of the following hashes

- 010101100000101001110011
- 010101100000101001110010
- 010101100000101001110001

⋮

- 110101100000101001110011

Figure 5. Neighbor Staged LSH

More generally, NSLH can be executed with any bit length of hash greater than 1. It does not necessarily be 24. The reason why we adopt 24 is to tune the performance and not to decrease the accuracy rate, like the followings.

1. The AFP database size will be more than 10 million. To optimize the number of calculating the hamming distance in the CS and ES steps, the average size of a bucket is desirable to be less than 1. To do so, the bit length of the hash should be equal to or greater than 24, because $2^{24} = 16777216 > 10$ million.
2. One of Araki's experiments [2] shows that the BER of AFP by HiFP2.0 was 15 % at most. Letting the BER of queries $e = 15\%$, the probability to successfully choose the correct bucket, which is leading to the correct AFP, is $(1 - e)^{24} + 24e(1 - e)^{23} = 10.6$ %. The probability to choose the correct bucket during the full of Staged LSH is $1 - (1 - 0.106)^{126} = 0.99999926 > 1 - 10^{-6}$, that is, almost 100%.

A rough estimation showed that NSLSH with 24-bit hash is almost $(2^{24} / 25) / 2^{13} = 81.92$ times as fast as the simple Staged LSH and the accuracy is almost same unless the BER of the query is less than 15%.

In fact, there is another concern about the required storage size. When hash is 13 bits, it is enough that there are 2^{13} pointers to the locations on the hash table, or 32768 bytes if the size of a pointer is 4 bytes. As for 24-bit hash, 2^{24} pointers or 67 MB are required. If we increase the bit length of hash more than this, the size of the pointers will not be negligible compared to the hash table and AFP database.

3.2 Hierarchy Staged LSH

HSLSH is another extension to use the hierarchical memory structure effectively. The idea is to focus on the popular songs. The hash table and database of 10 million AFPs occupies about 5 gigabytes each. It is inevitable to use a kind of storage. However, if there are frequently referred buckets and AFPs in the hash table and database, the performance could be faster by storing them into fast memories.

Figure 6 is how to store the hash table and database in two memories. The simple Staged LSH was not assumed to use multiple memories, so the naïve extension is that the hash table is stored in the fast memory as much as possible and the remaining part of the hash table and the full of the database are stored in the large memory. On the other hand, HSLSH stores the hash table and database of AFPs as many as possible, and stores the remaining hash table and database in the large memory.

Additionally, HSLSH first executes Staged LSH only with the fast memory, and then executes Staged LSH with the large memory only if no matching AFP is found in the fast memory. By doing so, HSLSH can omit to access the large memory if the query is an AFP from the popular songs.

Any cache algorithm could be thought, but we adopted a simpler method this time. HSLSH can be replaced with any more advanced cache algorithm, which may or may not follow the inclusion property.

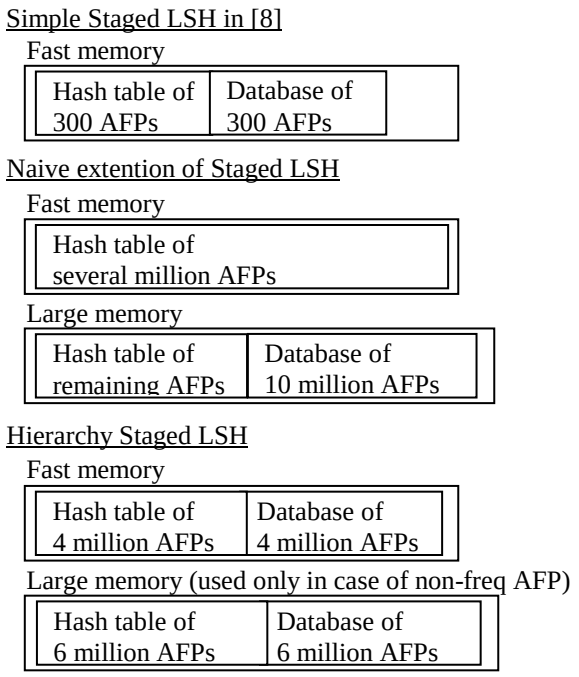

Figure 6. Hierarchy Staged LSH

3.3 Combination of two methods

Table 1 shows that there can be four implementations, depending on whether we adopt NSLSH and HSLSH or not.

Table 1. Abbreviations of four implementations

		NSLSH	
		disabled	**enabled**
HSLSH	**disabled**	Ⓢ	Ⓝ
	enabled	Ⓗ	ⓃⒽ

The ⓃⒽ in the table means the adoption of both NSLSH and HSLSH. It first executes NSLSH only with the fast memory, and then executes NSLSH with the large memory only if no matching AFP was found in the fast memory.

3.4 Implementations

Table 2 and 3 describes the FPGA evaluation board and its EDA tool which we used to implement the four methods.

Table 2. FPGA evaluation board

Parameters	Descriptions
Product Name	VC709 Connectivity Kit
FPGA	Virtex-7 XC7VX690T-2FFG1761C
DRAM	2 × 4gigabytes, DDR3
PCIe	PCIe 3 (x8)

Table 3. EDA tool for the FPGA

Environment Parameter	Descriptions
EDA	Vivado Design Suite - HLx Edition 2016.4
IP for DRAM	Memory Interface Generator 7 Series Ver. 2.4 (Rev. 1)
IP for PCIe	Virtex-7 FPGA Gen3 Integrated Block for PCI Express Ver. 4.1 (Rev. 1)

Table 4 is the main components of the desktop PC used as a large memory. It is accessed by the FPGA through PCIe to read the hash table and database. The CPU is only used to initialize the device driver, building the hash table and database and displaying the results of the experiments, so it is never contributed to the speedup of searching AFPs.

Table 4. Main components of the large memory

Parameters	Descriptions
Motherboard	Gigabyte MW50-SV0
CPU	Intel Xeon CPU E5-1620 v3 (4 cores)
Chipset	Intel C612
Memory	DDR4 33.6GB
I/O	PCI Express 3.0 x8, etc
OS	Ubuntu 16.04.1 Long Term Support
Kernel	4.4.0-62-generic (64 bits)

Figure 7 is the block diagram of our implementations. MFPS module is the key module and it executes Staged LSH or one of its extensions. It reads the hash table and database from fast or large memory through DRAM or PCIe.

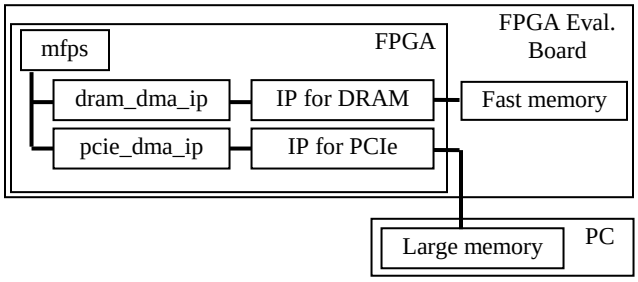

Figure 7. Block design of the top module

The module named pcie_dma_ip is a modification of the 7 Series FPGAs Integrated Block for PCI Express example designs, which is described in PG054 [7] written by Xilinx, to improve the performance. The main functions are executing Memory Read and Memory Write to the root complex and generating an interrupt. The other inputs and outputs to the Xilinx's IP for PCIe are almost same as the example design, except for BRAM for Base Address Register 0. The BRAM in our design is mainly used to be

informed the first physical addresses of the large memory in the desktop PC, allocated by the device driver. There are a few other uses of BRAM, such as activating the mfps module when the bit 0 of the address 0 is set.

The above system does not include any network interface. The query AFPs are sent from the desktop PC through PCIe. Our focus in this paper is not supporting various network protocols of TCP/IP but searching AFPs faster.

4. EXPERIMENTS

To evaluate the speedup of NSLSH and HSLSH, we carried out the experiments

4.1 Experimental Conditions

Table 5 shows the experimental conditions. The database was constructed by random numbers and its size is 10 million × 4096-bit = 5.12 gigabytes. The size of the hash table is almost same as the database, because it contains the pointers to 126 frames or 4032 bits (≈ 4096 bits) per an AFP if the size of a pointer is 32-bit. About the song popularity, we assumed the simple Zipf's law, that is, the probability of appearing n-th AFP as the query is $P(n) \propto 1 / n$. More specifically,

$$P(n) = \frac{\frac{1}{n}}{\sum_{i=1}^{10000000} \frac{1}{i}}$$

The number of trials is 1000 and the average was calculated for evaluations. The bit error ratio of queries varied from 0 to 25 % and added by random number. The threshold of CS and ES steps were 24 and 1024, respectively. They are one fourth of the frame and AFP size, so the system can allow up to 25 % BER of queries.

Table 5. Experimental conditions

Parameters	Descriptions
AFP Database	10 million 4096-bit random numbers
Song Popularity	Simple Zipf's law
# of Trials	1,000
BER of Query	0 – 25 %
Th_1	24
Th_2	1024

4.2 Searching Speed

Figure 8 shows the latency per query evaluated in the experiments. In section 3.1, we estimated the speedup ratio of Ⓝ over Ⓢ as 81.92. Actually, Ⓝ was 80.5 times as fast as Ⓢ, and Ⓗ was 4.26 times compared with Ⓢ. Totally, the method ⓃⒽ was 182.8 times at most (BER = 11 %) as fast as Ⓢ in the range of BER less than 15 %. As a result, it achieved the speed equivalent to the 1 Gbps network routers. It is said that the probabilistic strategies in the proposed methods were successful.

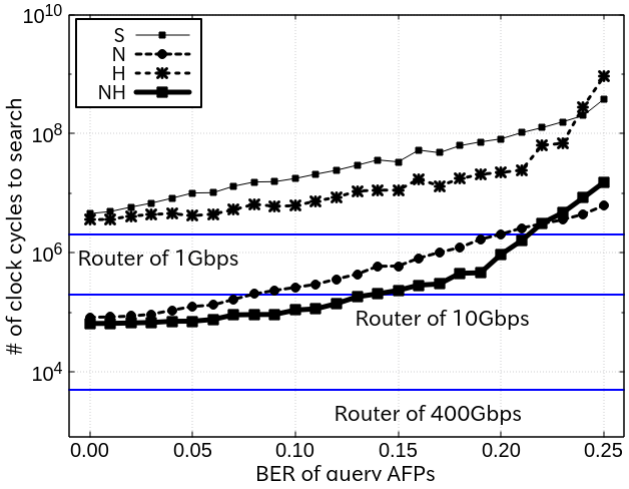

Figure 8. Latency in the experimental results

4.3 Accuracy Rate

Figure 9 shows the accuracy rate in the experiments. As far as the BER of queries was less than 15 % BER, all the proposed methods achieved 100 % accuracy rate as expected. On the other hand, when the BER was more than 15 %, the accuracy rates of Ⓝ and ⓃⒽ were worse. That is because the search scope of Coarse Search (CS) is small elite and CS unlikely looks at frames with big BER. Per Araki's experiments [2], however, the BER of AFP by HiFP2.0 was 15 % at most.

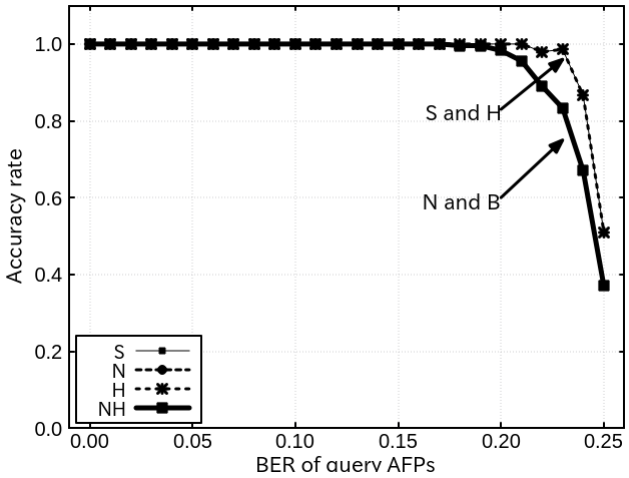

Figure 9. Accuracy rate in the experimental results.

4.4 Resource Utilizations

Table 6 shows the main resource utilizations of our implementations of the four methods. We successfully implemented all of them without error. The rates of increase of the resource are only up to 0.425 % (LUT).

Table 6. Resource Utilizations

Type	Utilizations				Avail-able
	Ⓢ	Ⓝ	Ⓗ	ⓃⒽ	
LUT	27,311	27,418	27,355	27,427	433,200
LUTRAM	3,446	3,446	3,446	3,446	174,200
FF	20,271	20,277	20,274	20,285	866,400
BRAM	41	41	41	41	1,470
IO	240	240	240	240	850
GT	8	8	8	8	36
BUFG	11	11	11	11	32
MMCM	4	4	4	4	20
PLL	2	2	2	2	20
PCIe	1	1	1	1	3

5. CONCLUSIONS

To realize easy and legitimate song sharing system on the Internet, we are challenging the speedup of AFP searches. Staged LSH in Yang's previous research [8] achieved almost 1 Gbps, but there was a problem that the experiment was carried out with only 300 songs.

In this paper, first, we added a large memory into the system to manage to use AFPs of 10 million songs. And then we extended Staged LSH with two probabilistic strategies and evaluated them by actual equipment.

NLSH adjusts the size and number of buckets to search in the Coarse Search process, based on the assumption that BER of AFPs tends to be small. It improved the searching speed significantly, up to 80.5 times. And it did not lead to worse accuracy rate in the range of 15 % BER, as we designed so.

HLSH effectively uses two kinds of memories based on the song popularity. It improved the searching speed a little, up to 4.26 times, without making the accuracy rate worse.

Totally, the combined method of NSLSH and HSLSH was 182.8 times as fast as the simple Staged LSH and achieved 1 Gbps with 100 % accuracy rate in the range of up to 15 % BER of queries.

The future work is developing an algorithm that dynamically exchanging data between the fast and large memories. It is preferable to improve HSLSH at the same time, so we are considering to implement some of famous cache algorithms such as Least Recently Used or Adaptive Replacement Cache by a moderate cache line size. Furthermore, we can also take advantage of the third memory named Block RAM. To carry out experiments about multiple cache algorithms, we should improve the development efficiency.

6. REFERENCES

[1] Apple. Use Apple Music in the Music app. https://support.apple.com/en-us/HT204951 (accessed on February 06, 2017).

[2] Araki, K., Sato, Y., Jain, V., and Inoguchi, Y. 2011. Performance Evaluation of Audio Fingerprint Generation using Haar Wavelet Transform. In *Proceedings of the International Workshop on Nonlinear Circuits, Communications and Signal Processing* (Tianjin, China, March 03, 2011). 380-383.

[3] Gionis, A., Indyk, P. and Motwani, R (1999). Similarity Search in High Dimensions via Hashing. In *Proceedings of the 25th Very Large Data Bases (VLDB) Conference*.

[4] Haitsma, J. and Kalker, T. 2002. A highly robust audio fingerprinting system. In *Proceedings of the International Symposium on Music Information Retrieval*.

[5] Gao, G., Liu, C. H., Chen, M., Guo, S. and Leung, K. K. (2016). Cloud-Based Actor Identification With Batch-Orthogonal Local-Sensitive Hashing and Sparse Representation. In *IEEE Trans. On Multimedia*. 18, 9 (Sep. 2016), 1749-1761. DOI= https://doi.org/10.1109/TMM.2016.2579305

[6] Hendrik, S. and Meinard, M. 2014. Accelerating Index-Based Audio Identification. In *IEEE Trans. on Multimedia*. 16, 6 (Oct. 2014), 1654-1664. DOI= https://doi.org/10.1109/TMM.2014.2318517.

[7] Xilinx. 7 Series FPGAs Integrated Block for PCI Express Product Guide (PG054). November 30, 2016.

[8] Yang, F., Sato, Y., Tan, and Y., Inoguchi, Y. 2012. Searching Acceleration for Audio Fingerprinting System. In *Joint Conference of Hokuriku Chapters of Electrical Societies* (September 01, 2012).