修士論文

楽曲電子指紋の時間的連続性に着目した省メモリ化と
データベースの実時間検索ハードウェアアクセラレーション

野本 健心

主指導教員 井口 寧

北陸先端科学技術大学院大学
先端科学技術研究科
（情報科学）

令和５年３月

Abstract

In resent years, with the development of an environment in which anyone can easily access the Internet, the market for digital content via networks is expanding, replacing CDs and other media in the distribution of music and other content. Furthermore, the increasing distribution of music via networks has transformed the characteristics of that distribution. In addition to the one-way distribution from the record producer who owns the master recording rights to the consumer, which has been done through e-commerce sites, two-way distribution is now widely used, such as the exchange of content between users who were previously on the consumer's side. This environment of quicker access to information than ever before has been created by the spread of social media and distribution services, and is widely used in the distribution of content where the freshness of information, such as current affairs and trends, is important. While the distribution of of content using P2P file sharing software such as Winny and large-scale distribution platforms such as Youtube and Spotify has increased the accessibility and speed of content dissemination, there is a high demand for speed in identifying rights holders, despite the increased complexity of content distribution management in policing copyright on the digital information with the potential for reproduction.

Regarding these problems, methods have been proposed for copyright management using Fingerprint technology. Fingerprinting is a technology that identifies songs using a compact representation of thousands of bits by analyzing signal data, such as energy transitions in each frequency band of a song, and extracting features. A system for copyright management using fingerprints could be implemented in routers of Internet providers or origin servers of distribution platforms to active content use while reducing the burden on the user side. In this case, a fast feature extraction method using hardware in signal processing is known for the generation of fingerprints. However, there are many difficulties in realizing highly accurate and high-speed search for music search using fingerprints, due to the nature of fingerprints, in which a fingerprint generated from a quality-degraded sound source is not completely identical to a fingerprint generated from the original sound source, but is similar to it.

Since songs exchanged over the network are often compressed using lossy methods such as mp3 for the purpose of saving network bandwidth, transfer time, and storage space, when such a song is input to the system, it is necessary to search for the nearest fingerprint in the database. In general, nearest neighbor element search has the problem of exploding computational complexity depending on the size of the database, known as the "curse of dimensionality". Thus, while music search

using fingerprints has robustness against quality degradation due to compression and other factors, the complexity of the search method makes it difficult to increase the speed. In addition, the scale of data covered by copyright management systems ranged from several hundred thousand songs to several million songs or more for each subscription music distribution service, indicating that high-speed identification is required for large data that will continue to grow in the future.

To address these issues, a probabilistic but fast nearest neighbor fingerprint search method using LSH(Locality-Sensitive Hashing), called Staged LSH, has been proposed. In addition, it has been found that offloading the method to an FPGA can achieve higher speeds by taking advantage of multiple arithmetic units in hardware. On the other hand, existing methods sometimes required multiple hash tables in the system in order to raise the search accuracy to a certain level. In such cases, the hash tables occupies several times as much memory space as the fingerprint database, which greatly limits the size of the manageable music fingerprint database in environment with only small memory space, such as embedded devices.

First, we improve the method of search fingerprints generated from songs by focusing on the temporal continuity of fingerprints, thereby improving the performance in the trade-off between accuracy, search speed, and memory space efficiency. When implemented on an FPGA device with the same accuracy and with parameters selected with priority on memory saving, the evaluation results showed that the hash table was reduced by approximately 74.99% and the speed was increased by approximately 7.03 times compared to the existing method in a database of 4 million fingerprints. In addition, in Neighbor Staged LSH with improvements over the Staged LSH, we achieved a speedup of about 109.79 times when parameters with comparable accuracy and memory space efficiency were selected. The results show that the proposed method can be widely adapted to LSH-based fingerprint search methods. The above results show that our method, which focuses on the temporal continuity of fingerprints, maintains robustness against bit errors in fingerprint search, improves memory space efficiency, and is effective in speeding up fingerprint search.

Next, we proposed and discussed a parallelization strategy to speed up the hardware implementation do the method. For the existing parallelization method focusing on independence among hash tables, we analyzed the issues in using global memory as a storage location for major elements, and proposed a new content-oriented parallelization method with a data structure based on content partitioning and a frame-oriented parallelization method that enables simultaneous search of multiple search elements. When each parallelization method was implemented and evaluated on a database of 4 million fingerprints for a small distortion fingerprint

input, the 4-parallel content-oriented parallelization achieved a speedup of approximately 2.16 times faster than the 1-parallel one. In addition, we have confirmed that frame-oriented parallelization is effective in terms of search speed in environments where memory accessible in parallel is limited and the input fingerprints have low similarity.

As described above, while maintaining equivalent accuracy, we improved memory space efficiency and increased speed to raise the performance level, thereby extending the size of the music a certain amount of time and and the size of the music fingerprint database that can be stored in limited memory space.

概要

近年、誰もが手軽にインターネットを利用できる環境が整備され、楽曲等のコンテンツの流通において、CD 等の媒体を通したコンテンツの流通の形態に置き換わり、ネットワークを介したデジタルコンテンツの市場規模が拡大している。さらに、ネットワークを経由した楽曲の流通が盛んになったことにより、その流通の特性についても変容している。これまで EC サイトなどを通じて行われていた、原盤権を持つレコード製作者から消費者といった一方向の流通形態に加え、それまで消費者側であったユーザー間のコンテンツのやり取りといった、双方向の流通が広く行われるようになった。これまで以上に素早く情報にアクセス可能な環境が、ソーシャルメディアや配信サービスの普及により整えられ、世相や流行など、情報の鮮度が重要なコンテンツの流通においても広く利用されている。Winny などの P2P ファイル共有ソフトや、Youtube や Spotify に代表される大規模配信プラットフォームを用いたコンテンツの流通では、コンテンツへのアクセス性や拡散速度が向上した一方、複製可能性を持つデジタル情報に対する著作権の取り締まりにおいて、コンテンツ流通管理の複雑性が増したことに加え、権利者の特定には高い速度の要求がある。

このような課題に対して、電子指紋技術を用いた著作権管理のための手法が考案されている。電子指紋は、楽曲の周波数帯域ごとのエネルギー遷移など、信号データを解析し特徴量を抽出することで、数千ビットのコンパクトな表現を用いて楽曲を識別する技術である。電子指紋を用いた著作権管理のためのシステムは、ユーザー側の負担を小さくしながら、コンテンツ利用の活性化を図るため、インターネットプロバイダのルータや、配信プラットフォームのオリジンサーバなどに実装することが考えられる。その際、指紋の生成については、信号処理においてハードウェアを用いた高速な特徴量の抽出手法が知られている。しかし、指紋を用いた楽曲検索については、品質劣化した音源から生成される指紋が、元の音源から生成された指紋と完全に同一のものとはならず、類似した指紋となるといった、電子指紋特有の性質があるため、高精度で高速な検索の実現には課題が多い。

ネットワークを介してやり取りされる楽曲は、ネットワーク帯域や転送に掛かる時間、保存領域の節約などの目的で、mp3 に代表する非可逆的な手法により圧縮されていることが多いため、システムに対して、このような楽曲が入力した時、データベース内の指紋から最近傍の指紋の検索を行う必要がある。一般に、最近傍の要素検索には「次元の呪い」と呼ばれるデータベース規模に応じた計算量の爆発的な増加の課題がある。そのため、電子指紋を用いた楽曲の検索では、類似指紋の検索により、圧縮などによって品質劣化した楽曲に対する頑健性を備えている一方、探索手法の複雑化により高速化が容易ではない。また、著作権管理システムの対象となるデータ規模は、各定額音楽配信サービスを見ても、数十～数百万楽曲以上となっており、今後も増大する大規模なデータに対して、高速な識

別が要求されることが分かる。

このような課題に対して、Staged LSH と呼ばれる LSH（局所性鋭敏型ハッシュ）を用いた、確率的でありながら高速な最近傍指紋の検索手法が提案されている。加えて、手法を FPGA にオフロードすることでハードウェア上の複数の演算器を活かした高速化が達成可能であることも分かっている。一方、既存の手法では、検索精度を一定の水準に引き上げるため、システムにおいて多数のハッシュテーブルを要求することがあった。このような場合、ハッシュテーブルが指紋データベースに対して数倍程度のメモリ空間を占めるため、組み込み機器等の小規模なメモリ空間しか持たない環境で、管理可能な楽曲指紋データベース規模に大きな制限を与えていた。

このような、楽曲から生成した電子指紋の検索手法について、指紋の時間的連続性に着目した改良を行うことで、精度、検索速度、メモリ空間効率のトレードオフにおける性能の底上げを行った。評価の結果から、精度を同等とし、省メモリ化を優先したパラメータを選択を選択したうえで、FPGA デバイス上に実装した時、400 万個の指紋データベースにおいて、既存手法と比較して、約 74.99%のハッシュテーブルの削減と、約 7.03 倍の高速化を達成した。また、Staged LSH に対して改良を加えた隣接 Staged LSH において、同等の精度およびメモリ空間効率を持つパラメータを選択した場合、約 109.79 倍の高速化を達成し、提案手法が LSH を基本とした指紋の検索手法へ広く適応可能であることを示した。以上から、指紋の時間的連続性に着目した本手法が、電子指紋検索のビットエラーに対するロバスト性を維持し、メモリ空間効率の向上ならびに、高速化において有効であることが分かった。

また、ハードウェア上に実装した手法について、並列化による高速化の考案および考察を行った。既存のハッシュテーブル間の独立性に着目した並列化手法について、グローバルメモリを主要な要素の格納場所として用いる際の課題を分析し、新たにコンテンツの分割によるデータ構造を持ったコンテンツ指向の並列化手法と、複数の検索要素の同時処理を可能とするフレーム指向の並列化手法を考案した。それぞれの並列化手法を実装し、歪みの小さな指紋入力に対して 400 万個の指紋データベース上で評価を行ったところ、4 並列のコンテンツ指向の並列化では、1 並列のものと比較して約 2.16 倍の高速化を達成した。加えて、並列にアクセス可能なメモリが制限される環境において、入力される指紋の類似度が低い場合、検索速度の面でフレーム指向並列化が有効であることを確かめた。

以上のように、同等の精度を維持したまま、メモリ空間効率の向上と高速化を行い、性能の底上げをすることで、一定時間内に検索を可能とする楽曲指紋データベース規模の拡張と、限られたメモリ空間において管理可能な楽曲指紋データベース規模の拡張を行った。

目 次

図 目 次

表 目 次

アルゴリズム目次

ソースコード目次

第1章　導入

1.1　研究の背景

インターネットを介したデジタルコンテンツの配信は場所、時間を問わず断続的に行われており、これらのコンテンツに対して誰もが手軽にアクセス可能な環境が整っている。文献 [1, p.77] によると 2017 年におけるデジタルコンテンツとしての音楽の世界的な流通状況は、ネットワークを経由したデジタル流通が 55%以上と市場規模が最も大きく、CD 等の媒体を利用した市場が斜陽化し、各種ストリーミング配信サービスに移行しつつある現状を見ると、今後もさらにネットワークを介した音楽の流通が拡大していくものと容易に予想ができる。また、これまで受け手であったユーザーを含めた不特定多数が配信可能な双方向のサービスの拡大により、ユーザー間のコンテンツの受け渡しや、独自コンテンツの配信なども容易となっている。一方、誰もが簡単にコンテンツをやり取り可能なサービス上は、著作物の無断アップロードなど、違法利用の温床となることも少なくなく、無秩序な著作物が氾濫する状況が生じている [2]。加えて、単純に情報を発信する母数が大きくなったことで、人手による著作権の取り締まりに限界が見え始めている。

このような問題に対して、電子指紋（FPID：Fingerprint Identification）技術を用いた著作権管理システムによるデジタル著作物の権利を保護する試みが行われている。楽曲を対象とした電子指紋技術では、周波数帯域ごとのエネルギー遷移など、信号データを解析し特徴量を抽出することで、数千ビットのコンパクトな表現を用いて著作権の特定が行われる。

図 1.1 のような電子指紋技術を用いたデジタル著作権管理（DRM：Digital Rights Management）システムは、配信プラットフォームのコンテンツが格納/配信されるサーバー上や、P2P アプリケーションの経由するルータ上での運用などが考えられる。ここで、大規模な配信プラットフォームである Youtube[*1] に代表されるような、1 分間におよそ 500 時間程度のコンテンツがアップロードされ続ける[*2] 大規模なプラットフォームや、ライブ配信サービスや P2P でのファイル交換において、ブロードバンド上で高速に通信、交換されるコンテンツに対して実時間での楽曲の特定を行うことを仮定すると、高速な楽曲の識別が要求されることは明らかである。しかし、電子指紋を用いた識別処理において、必ずしも同じ楽曲から完全

[*1] https://www.youtube.com/
[*2] https://www.domo.com/data-never-sleeps

に同一の電子指紋が生成できるとは限らないことが、検索における高速化の障害となる。ネットワーク経由で共有される楽曲ファイルは、ネットワーク帯域や転送にかかる時間、保存容量の節約などの目的から、ユーザーによって mp3 に代表される各種手法により不可逆的に圧縮が行われていることが多い。このような楽曲ファイルから生成される電子指紋は、圧縮前の楽曲から生成された電子指紋と比較して歪みを持つため、楽曲指紋データベースに格納した電子指紋の最近傍にあたる、類似した指紋の検索アルゴリズムが要求される。

既存の手法として、局所性鋭敏型ハッシュ（LSH:Locality Sensitive Hashing）とハッシュテーブルを用いた確率的な探索手法をハードウェアアクセラレータへ適応した研究では、一定の以上の高速化を達成した一方、精度維持のための、ハッシュテーブル増加による、メモリの圧迫に目に余るものがある。数十～数百万曲以上の大規模なデータを対象とする著作権管理システムにおいて、メモリ空間効率悪化の課題はデータベース規模が大きくなるほど、組み込み機器等のメモリ制約などにおける課題として顕在化する。また、DRAM や SRAM といったメモリはサイズやコストの観点から制限が多く、安易に増設できないことも少なくないため省メモリ化はより重要視される点である。

上記のような課題を解決し、電子指紋技術によってデジタルコンテンツの著作権管理を行うシステムを構築できれば、既に流通する楽曲も含め、ユーザー側の負担を最小限に抑えながら、著作権の所在をシステムによって容易に明らかすることが可能となる。これにより、コンテンツ利活用のさらなる活性化が期待できる点に本研究の意義が認められる。

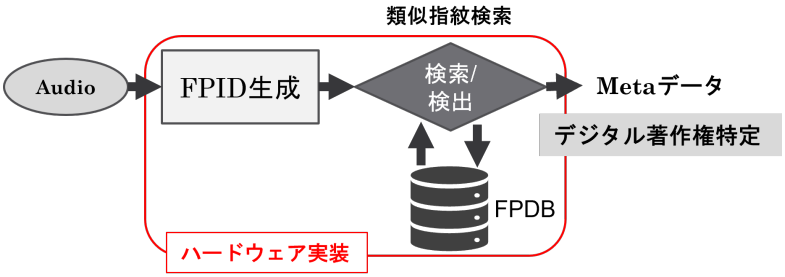

図 1.1: 電子指紋を用いたデジタル著作権管理システム概要

1.2 研究の目的

電子指紋を用いた楽曲識別には局所性鋭敏型ハッシュを用いた確率的な探索手法が用いられ、一定以上の精度を指紋の検索において保障するため、ハッシュテーブルに楽曲データベースのおよそ数倍のメモリ領域が要求される場合がある。しかし、流通する楽曲は各定額音楽配信サービスを見ても数百万楽曲以上となる [3, p.10] ため、このような大規模な楽曲データベースに対して、検索に必要なこれらの要素が FPGA デバイス等のメモリサイズの上限に達する可能性がある。すると、

上限を超える要素についてはSSDなどの外部ストレージへ格納するほかなく、デバイスに搭載されているグローバルメモリ等に比べ、要素の読み出しのためのレイテンシが大きいため、検索の高速化に対する影響が大きく、大幅な速度の低下が懸念される。

既存のハードウェアを用いた高速な検索システムを外部のストレージにできる限り頼らず、大規模な楽曲データベースへ適応させるため、第一に検索機構におけるハッシュテーブル削減の要求がある。そこで、新たに考案する楽曲から生成された指紋の時間的な連続性に着目した手法により、データ構造および格納、検索手法の改良を行うことで、精度、速度、メモリ空間効率間のトレードオフにおける性能の底上げを行う。これにより、検索精度を一定以上で同等に維持したまま、省メモリ化を達成し、組み込み機器等における小規模なメモリ空間への適応など、システム利用の幅を拡大することができる。また、省メモリ化に加えて、検索の高速化についても同時に達成することで、要求される数ミリ秒以内に処理可能な楽曲データベース規模を拡張可能であることも示す。加えて、LSHを用いた手法について、ハッシュテーブルサイズと精度のトレードオフの関係を底上げした既存の手法へ、本論文で考案した手法を上乗せすることで、提案手法がLSHを基本とした指紋の検索手法へ適応可能であることを検証する。

また、ハードウェア実装における回路の並列化による高速化戦略について、既存の、ハッシュテーブル間の依存性がないことに着目した、ハッシュ関数指向並列化の問題点について言及し、新しい高速化を目的とする並列化手法による実装の評価および考察を行う。

以上、LSHを用いた電子指紋検索の手法の改良と、新しい並列化による高速化手法の2点について、これらをFPGAデバイス上へ実装し、手法の評価および考察を行う。

1.3 本文の構成

第2章では、電子指紋技術の要旨とそれらを用いた著作権管理技術の要件について議論する。また、指紋の生成と検索技術についての関連研究とその分析および、考察を行う。第3章では、指紋の時間的連続性に着目した提案手法の概要と、評価に用いるパラメータの決定方法について議論し、関連研究との比較した場合の本手法の立ち位置について考察する。第4章では、手法をFPGAデバイスへ実装する際の、並列化による高速化戦略について新たな手法を提案し、それぞれの手法の意図および効果について、関連研究を踏まえたうえで考察する。第5章では、まず、第3章で示した提案手法について、手法の効果を評価する。次に第4章で示した各並列化戦略について、1並列および他手法と比較した場合の手法の利点、有効性について評価および議論を行う。最後に第6章では、結論および、今回の研究を通じて考察した今後の課題についてまとめている。

第2章　電子指紋技術の関連研究とその分析

2.1　はじめに

本章では、まず電子指紋技術の概要と著作権管理システムにおけるその位置付けについて簡単に示す。次に、指紋の生成と指紋を用いた楽曲の検索における関連研究について紹介する。最後に、今回対象とする指紋を用いた楽曲の検索手法についての分析と課題の考察を行う。

2.2　電子指紋技術の概要

テキストベースの楽曲の検索は、比較的容易に行うことができるため、多数の楽曲の歌詞を保有したプラットフォームなどがすでに普及している [4][5]。しかし、コンテンツベースの高精度な楽曲検索技術は研究段階であり、連続する音符間のピッチ差などを利用した文献 [6] などの手法が考案されている。中でも、電子指紋技術はコンテンツ ID とも呼ばれ、ハミングによる楽曲検索 [7] や画像や動画などの識別などにも利用されている [8]。本文では、楽曲の電子指紋を用いた著作権管理技術に特に注目する。

2.2.1　電子指紋と電子透かし

著作権保護のための楽曲識別に用いられる手法として、電子透かしと呼ばれる技術がある。電子透かしは、著作権の管理や不正利用の追跡などに主に用いられる技術として注目されている。用途と使用されるシステムが重なる点からみると、電子指紋技術と類似した手法のように見えるが、使用にあたって以下のような要求がある [9, p.2]。

- **透かしが記録されたコンテンツの品質をほとんど損なわないこと。**

  電子透かしは、楽曲から見ると一種の雑音信号のため、透かしによる品質への影響を最小限にする。

- **コンテンツの編集や圧縮、フォーマット変換を含む各種改変に対して透かしが消えないこと。**

  使用形態によるもの、あるいは、悪意のある者によるコンテンツの改変に対して、電子透かしがある程度の耐性を有することが要求される。

- **透かしの読み出しは許可された者のみが可能であること。**

  一般に透かしの読み出しが可能であれば、消去も可能であるため、不正な複製などの行為を抑制している場合には、著作権者に準ずる者のみが透かしを読み出す鍵を所有している必要がある。

電子透かしを利用した手法では、コンテンツそのものにメタデータを与える必要があるため、ノイズに敏感なオーディオには適用が困難であったり、メタデータそのものに影響を与えるような、コンテンツの圧縮による品質劣化に対して脆弱な場合がある。

一方、電子指紋技術ではコンテンツそのものにメタデータを与える必要がないため、ノイズに敏感なオーディオに対しても用いることができる。また、コンテンツそのものからダイジェストを生成することから、メタデータに基づく手法と比較すると、オーディオ圧縮などによる品質劣化に頑健である点が特徴として挙げられる。さらに、電子指紋技術は電子透かしと比べて事前の前処理なしで楽曲が識別できる利点があるため、すでに流通している楽曲に対しても有効である点が、不特定多数がアップロード可能なプラットフォームといった環境において重宝される。

ここまで、楽曲を特定するための類似した 2 手法について比較してきたが、電子指紋は電子透かしに必ずしも対立する技術という立ち位置にあるわけではなく、電子透かしに対して補完的な技術として用いられることもある [10]。

2.2.2 電子指紋を用いた著作権管理システム

不特定多数のユーザーを主体としたコンテンツの配信環境が整っているにも関わらず、楽曲などの著作物の使用には、権利などの都合上、事前の申請や権利者への許諾が必要であるため、だれもが気軽にコンテンツを利用できる状況と言えない。電子指紋により高速に楽曲を識別することが可能な機構を、ルータなどのネットワーク機器や配信サーバー上に組み込むことができれば、デジタル著作物の権利の所在をリアルタイムに明らかにしつつ、適正なコンテンツの流通をコントロールすることができ、コンテンツの利活用が促進されることが期待される。

そこで、電子指紋を用いた著作権管理システムでは、図 2.1 や図 2.2 に示すように、P2P 通信における楽曲ファイルの交換や、配信サーバー上にアップロードさ

れる楽曲に対して権利者の特定を行う用途が考えられる。また、近年のラジオや動画形式のライブ配信ストリーミングサービスなどの、不特定多数のユーザーが簡単にインターネット上でのライブ配信が可能となっている状況に対しても、リアルタイム性のある著作権管理システムの要求が見込まれる。

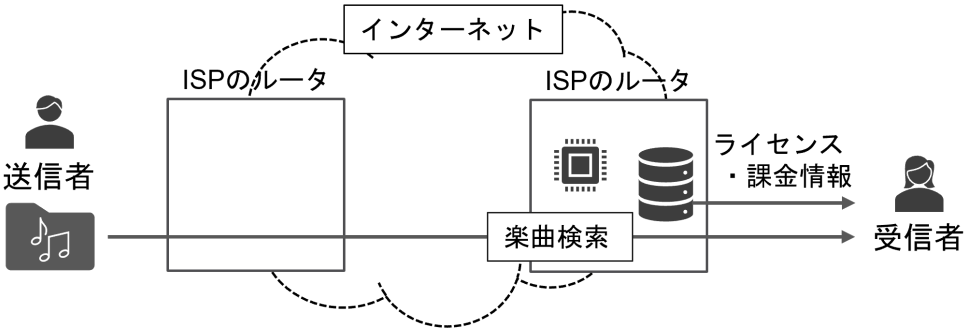

図 2.1: P2P ファイル交換ソフトにおける著作権管理システム例

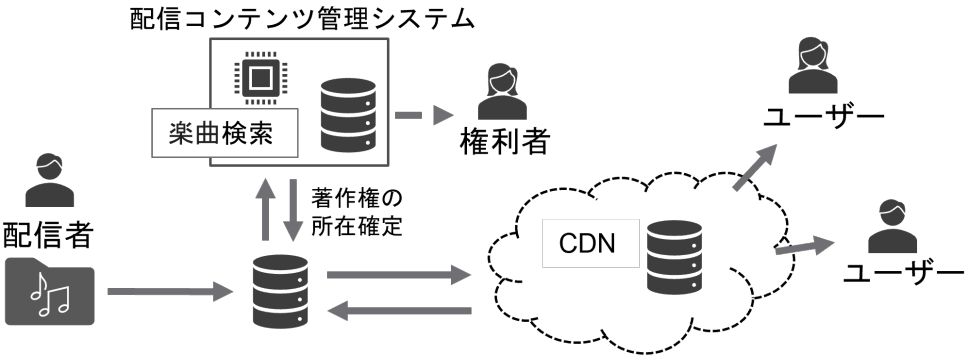

図 2.2: 配信プラットフォームにおける著作権管理システム例

加えて、NFT などの近年注目されているコンテンツ管理のための技術においても、海賊版や無許可での権利使用問題 [11, p.133] などの課題がある。このような他者の権利所有物の可能性があるデジタル著作物などについて、サービス運営側による監視機能として意図しないコンテンツの流出防止などを行う技術としても応用可能である。

以上のような著作権の管理システムに要求される処理速度について考察すると、$10Gbps$ で接続されたブロードバンド上を例に挙げると、頻繁に用いられる圧縮率 $256kbps$ で圧縮された 3 分の楽曲のデータ量は約 $46Mbit$ となり、転送速度は約 $4.6ms$ であることが分かる。また、1 分間に 500 時間分のコンテンツがアップロードされる大規模なプラットフォーム上で、1 曲 3 分の楽曲が毎分 1 万楽曲アップロードされていると仮定すると、1 曲あたりに許容される処理時間は約 $6.0ms$ となる。以上から、大規模な楽曲指紋データベースを対象とした、数ミリ秒程度の楽曲識別速度の要求が予測されることが分かる。

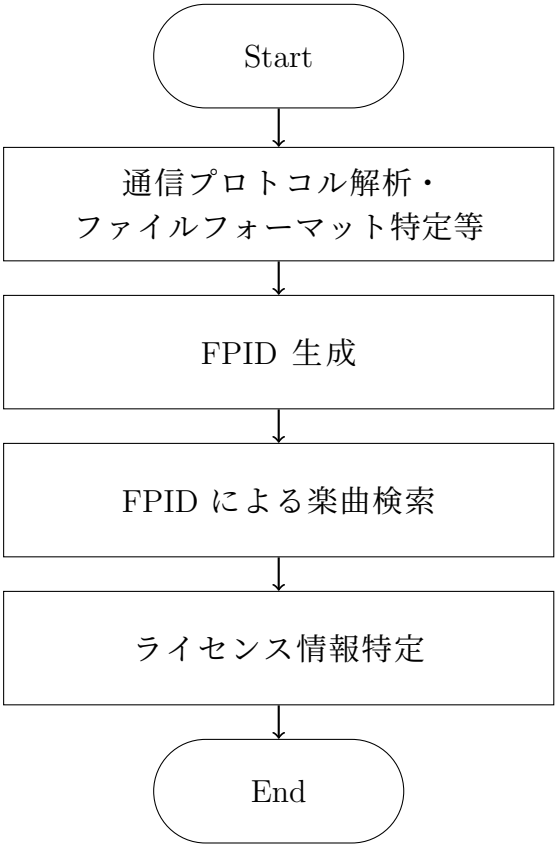

図 2.3: 電子指紋（FPID）を用いた著作権管理システムの処理フロー

著作権管理システム上の大まかな処理の流れは図 2.3 のようになる。楽曲がシステムに入力することで最終的に著作物のライセンス情報を特定し、ユーザーに対して課金情報を請求したり、適切な権利者に権利を割り当てる処理を行う。処理フロー中で、特に電子指紋の生成と検索については研究の段階で様々な手法が考案されている。以下の節で今回実験に用いた関連のある手法について紹介する。

2.3 電子指紋の生成

楽曲の特徴量を抽出し電子指紋を作成する手法として、文献 [12] により提案された手法では、オーディオ信号の周波数帯域などの特徴量を解析することで、オーディオ内容の知覚部をコンパクトに表現し、信号加工処理によるオーディオ劣化に頑健なコンテンツのダイジェストとして楽曲電子指紋を作成した。一方、特徴解析過程における、離散フーリエ変換（FFT）などのデジタル信号処理部について、ソフトウェア上での実装では、広域ネットワーク上で流通する楽曲ファイルの実時間での管理・識別のための十分な速度を達成することは困難であった。

そこで、CPU での実装に比べて、デジタル信号処理に長けたハードウェアを用いた電子指紋生成の手法が考案されている。ハードウェア実装において離散フー

リエ変換は、乗算器や大規模メモリを必要とし、ボトルネックとなるため、文献[13]では、楽曲からの特徴量の抽出に離散ウェーブレット変換（DWT）のサブバンド分解を用いたHiFP（High speed Audio Fingerprint Generation System）と呼ばれる手法を使用して楽曲電子指紋の生成が可能であることを示した。

アルゴリズム 2.3.1 DWTのサブバンド分解のアルゴリズム

```
DWT(
wav[] ← Inputsignal,                          ▷ 入力信号（楽曲データ）
n ← Numberofsamples,                          ▷ 入力サンプル数
m ← Numberofsamplesofoutput){                 ▷ 出力サンプル数
while n ≥ m do
    n = n/2;
    for i = 0 to n do
        Hi[i] = (wav[2 × i] − wav[2 × i + 1])/2;   ▷ 移動平均
        Lo[i] = (wav[2 × i] + wav[2 × i + 1])/2;   ▷ 算術平均
    end for
    wav[] ← Lo[];
end while
return(Hi, Lo);
}
```

アルゴリズム 2.3.2 HiFPによるFPID生成のアルゴリズム

```
HiFP(
wav[] ← PCMdata(131, 072sample)){             ▷ 2.97秒相当のサンプル
n ← 131, 072;                                 ▷ 入力サンプル数
m ← 4, 096;                                   ▷ DWTサブバンド分解の出力サンプル数
Hi[], Lo[] ← DWT(wav[], n, m);                ▷ DWTサブバンド分解
for k = 0 to m − 2 do
    tmp_fpid = Hi[k] − Hi[k + 1];             ▷ 特徴の抽出
    if tmp_fpid > 0 then                      ▷ ビット割り当て
        FPID[k] = 1;
    else
        FPID[k] = 0;
    end if
end for
FPID[m − 1] = 0;                              ▷ 4096ビット目に0を挿入
return FPID;
}
```

アルゴリズム 2.3.1 で示すDWTのサブバンド分解アルゴリズムでは、PCM 音源から取得した 32 サンプルを 5 段の回路で処理し、前半 4 段で算術平均を用いて低周波数成分を、最後の 1 段で移動平均を用いて高周波数成分の特徴を抽出することで、人の可聴域の時間周波数成分を取得している。これにより、人の可聴域外の周波数成分がカットされる各圧縮手法に対して、ロバスト性を向上させることができる。最後にアルゴリズム 2.3.2 の 7,8 行目において、前後の要素の差分を取ることでビットの割り当てを行い、4096 ビットの指紋を生成している。

Haar ウェーブレット変換の基底関数を整数の加減算とビットシフトのみで構成可能な点や、高い並列性と処理のパイプライン化によるスループットの向上を達成し、FPGA 上の実装により、最大 70.3*Gbps* のネットワークに適応可能であることを示した。また、既存手法と比較して楽曲の劣化に対しての頑健性の向上も達成し、検索における信頼性についても向上した。

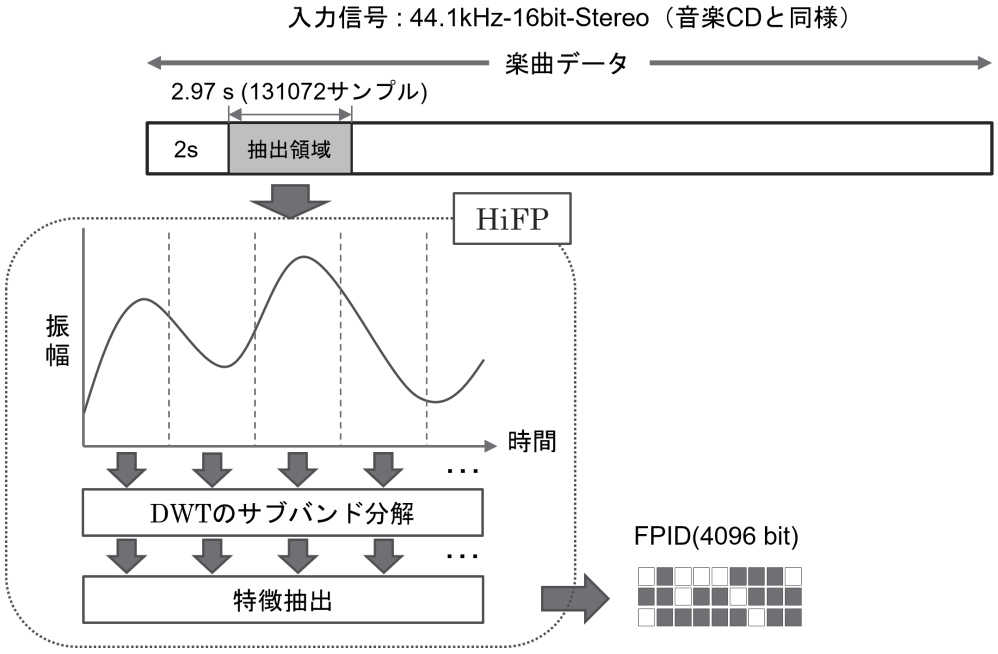

図 2.4: 楽曲の特徴抽出および電子指紋の生成概要

今回の電子指紋による楽曲の検索における評価おいても、図 2.4 のように無音区間を考慮した楽曲の開始から 2 秒を除いた、3 秒程度の区間から生成した電子指紋を、その楽曲を識別するためのダイジェストとして使用することを仮定している。

2.4 電子指紋の検索

多くの場合において、ネットワークを介してやり取りされる楽曲はネットワーク帯域の節約や伝送速度の高速化、ストレージの節約などのために、mp3 に代表

される圧縮手法で元の PCM 音源から、人の知覚の性質を利用した不必要なデータの除去が行われている。また、強力な圧縮手法の大半は非可逆的な手法であるため [14, p.169]、圧縮解除を行った場合にも、オリジナルの情報を正確に得ることはできない。このような圧縮後のデータから、2.3 節の手法を用いて生成された楽曲電子指紋は、同一楽曲であったとしても元の楽曲データから生成された指紋と完全に同じものとはならず、限りなく類似した指紋となる。

指紋同士の類似度を比較するための尺度としては、ハミング距離を用いることができる。ハミング距離は、比較する指紋同士の排他的論理和の後、1 の立っているビットを数える、ポップカウントを行うことで求めることができる。求めたハミング距離は、同一楽曲から生成された指紋と比較した場合のビットエラーとなる。指紋あたりに含まれるビットエラーの割合をビットエラー率 (BER: $BitErrorRatio$) といい、楽曲の識別子を $index$、ある楽曲を M_{index}、指紋を FP とすると式 (2.1) のように計算できる。ここで、4096 は 2.3 節の指紋の生成手法によって生成した指紋 1 つあたりの大きさである。

$$BER = \frac{1}{4096} \sum_{i=0}^{4096-1} FP\left(M_1\right)[i] \oplus FP\left(M_2\right)[i] \tag{2.1}$$

ある要素に最も近似した要素をデータ集合から探索するための、最も自然な手法であるブルートフォース戦略では、検索クエリの指紋と、指紋データベース内の指紋を 1 つずつ比較し、ハミング距離が最小となる要素を探索する。このような探索手法では、n 個の指紋が格納されたデータベースを想定すると、1 度の検索毎に n 回の指紋同士のハミング距離計算および、閾値や最小ハミング距離との比較を行う必要がある。このような、原始的な探索手法を用いることで、確実に最近傍の要素を発見可能である一方、流通する楽曲は数十～数百万曲以上に上ることを考慮すると、このような非常に大きなデータ集合に対しては、探索効率が悪いと言わざるを得ない。そこで、最近傍の電子指紋の検索要求に対して、ハッシュ法を応用した探索、比較数の削減を行う手法が用いられている。

2.4.1 Staged LSH

著作権管理システム上での電子指紋を用いた楽曲の識別を想定すると、圧縮等による、楽曲の劣化により、生成された指紋に多数のビットエラーを含む場合でも、対象の楽曲を発見する要求が見込まれる。また、クエリに該当する楽曲の特定に、高精度な識別を保証したい。そこで、高い精度の要求に対し、柔軟に適応可能な手法として、LSH（局所性鋭敏型ハッシュ）を用いた探索手法が用いられている。LSH は文献 [15] で提案されていた、最近傍問題（nearest neighbor problem）の「次元の呪い」を解決するための、近似最近傍問題のための手法を、文献 [16] で改良した、確率的に類似した要素を発見することが可能な手法である。これを用

いた、クエリに対して最近傍な指紋を探索するための、高速な手法として文献 [17] において Staged LSH と呼ばれる手法が提案されている。以下では、Staged LSH の概要について紹介する。

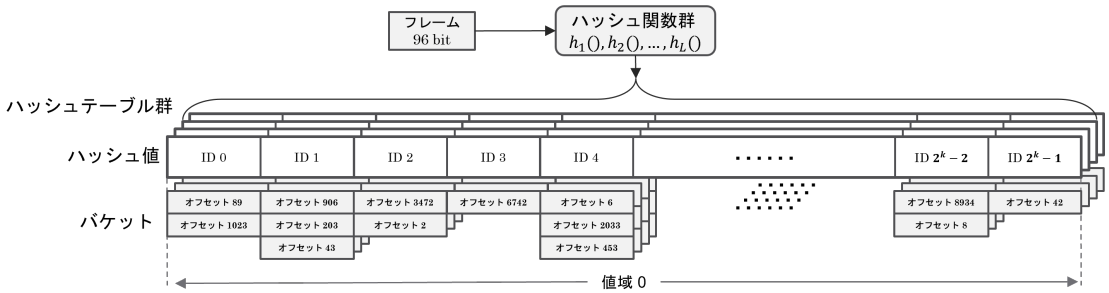

図 2.5: ハッシュ関数とハッシュテーブルの例

LSH を用いた最近傍の指紋検索においても、単純なハッシュ関数を用いた手法を同じように、図 2.5 のような、ハッシュ関数とそれに対応するハッシュテーブルの組を用いる。ここで、ハッシュ法を用いた同一要素の探索と比較した場合、類似要素検索際の特筆すべき特徴としては、図のように要素の格納および検索において複数のハッシュ関数と、それぞれのハッシュ関数に対応する、複数のハッシュテーブルを用いることが必要とされる場合がある点にある。手法自体が確率的な探索手法である、Staged LSH 手法では、高い検索精度が求められる場合、複数のハッシュ関数を用いることで精度の調節を行うことができる。当然、ハッシュテーブル数が増えると、メモリに格納する要素の数も増加するため、検索精度と要素を格納するメモリ使用量には、トレードオフがあることが分かる。精度とメモリ空間効率の関係の詳細については、2.5 節以降で後述する。

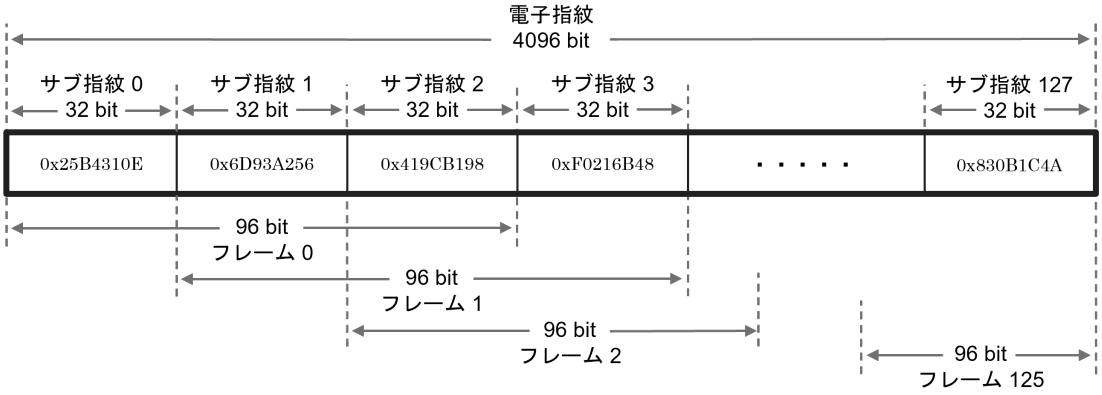

図 2.6: サブ指紋とフレームの概要

Staged LSH では電子指紋の検索において、図 2.6 で示すように、指紋からフレームと呼ばれる 96 ビットの 3 つのサブ指紋から構成された要素を取り出し、探索 1

回あたりの検索単位としている。これにより、各指紋検索の度に最大 126 個のフレームを用いた検索処理を可能とする。ここで、各指紋から取り出したフレーム先頭を指す指紋データベース配列先頭からのオフセットが、ハッシュテーブル内のバケットに格納される要素として用いられる。

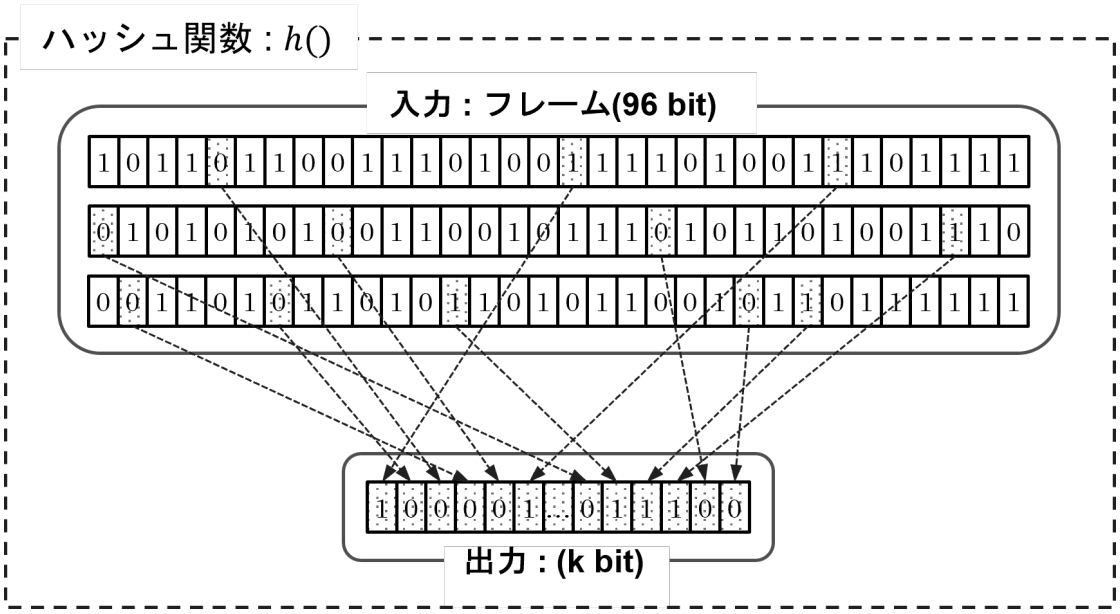

図 2.7: ハッシュ関数の例

また、格納および検索には、図 2.7 のようなハッシュ関数を用いる。これは、入力された 96 ビットのフレームから、ランダムに選出した特定のビット位置を k ビット取得する、ビットサンプリングによって構成されている。この方法により、構成されるハッシュ関数をハードウェア上に実装することで、乗算や加算などの演算を必要とせず、ビットを保持するレジスタ間の配線の切り替えのみで構成可能である利点がある。ただし、精度や速度の要求変更に対する、各種パラメータの変更要求に対して、ソフトウェア実装の場合と比較して柔軟性を失う。

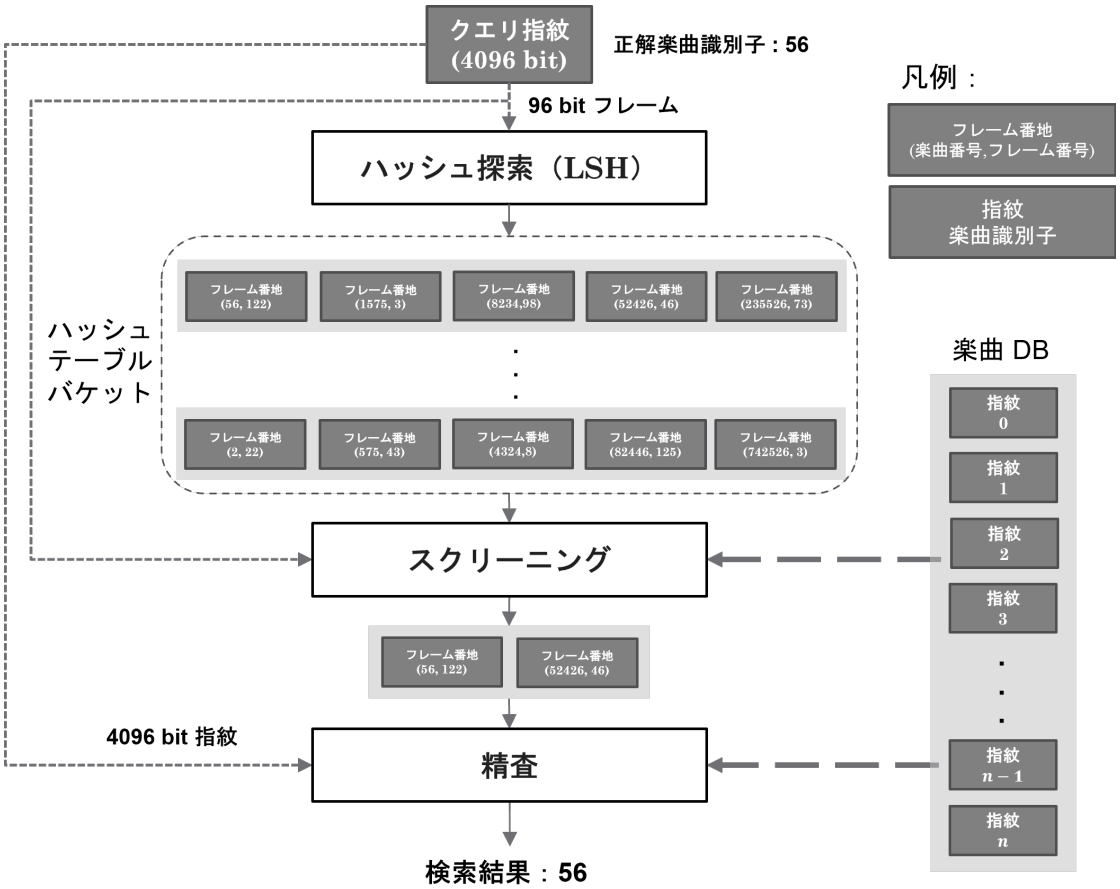

図 2.8: Staged LSH による検索処理フロー

Staged LSH を、単純な LSH を用いた探索手法と比較した際の大きな利点は、検索処理時間に占める割合の大きい、4096 ビットの指紋すべてに対するハミング距離計算の回数を削減したことにある。図 2.8 で表されるように、検索処理は主に 2 つの段階に分けて探索範囲の縮小を行い、3 つ目の段階の精査において最終的な楽曲識別子の判定および特定を行っている。各段階における処理の詳細は以下の通りである。

- **ハッシュ探索（*LSH*）**

  入力したフレームからハッシュ値を計算し、ハッシュ値から対応するハッシュテーブル内の一意に決まるバケットを特定する。加えて、バケットの探索範囲を求める。

- **スクリーニング**

  バケット内に格納されている各フレーム先頭位置の情報から、フレームの特

定、当該フレームの読み込みを行い、検索対象の指紋から取り出された入力フレームとハミング距離計算を行う。ビットエラーの数が閾値以下の要素を次の精査処理へと移行する。

- **精査**

  スクリーニングを通過した指紋に対して、検索対象の指紋と 4096 ビットすべてを対象としたハミング距離の計算を行い、閾値以下で、かつバケット内で最もビットエラー数の小さい指紋を特定し楽曲の識別を行う。

上述したように、探索においてフレームによる探索範囲の荒い絞り込み段階を設けることで、指紋すべての比較に移行する要素の数を削減することが可能となり、4096 ビットの指紋すべての比較だけを行う場合に比べて、速度の向上を測ったことが Staged LSH の功績である。この手法では、段階的な比較処理を設けない場合と比較して最大 45 倍、最悪でも 13 倍の高速化を達成した。

前段階のスクリーニングでは、指紋から取り出される、一定以上大きな単位要素を対象とするほど、指紋すべてを対象とする比較処理の回数が少なくなるため、高速化が望める。しかし、あまりに大きなものを選びすぎると、前段階のハミング距離計算自体に時間が掛かかったり、処理のための回路が大型化する可能性がある。そこで、既存の手法では 96 ビットのフレームを前段階における処理の対象としていた。今回の提案手法の評価に用いる実装においても既存の手法に準拠し、96 ビットのフレーム単位で検索を行った。Staged LSH の 1 ハッシュテーブルあたりの探索アルゴリズムはアルゴリズム 2.4.1 のようになる。

アルゴリズム 2.4.1 Staged LSH による検索処理部

```
1: for one frame t ∈ query fingerprint Q do
2:     HashID ← process t with the Hash Function;        ▷ ハッシュ値の生成
3:     for i ∈ HashTable do
4:         if HashID == HashTableEntry[i].HashID then
5:             for j ∈ Bucket do
6:                 if CoarseSerch(t, BucketEntry) ≤ Threshold_1 then
7:                     P ← FPDB, based on the bucket entry;
8:                     if ExactSearch(P, Q) ≤ Threshold_2 &P < BestMatch
   then
9:                         BestMatch ← P;
10:                    end if
11:                end if
12:            end for
13:            return BestMatch;
14:        end if
15:    end for
16: end for
17: return "NO_MATCH";
```

2.4.1.1 隣接 Staged LSH

文献 [18] によって提案された、Staged LSH のハッシュ関数部を改良した隣接 Staged LSH という手法がある。これは、類似した指紋からビットサンプリングによって得られたハッシュ値は、完全に一致する、あるいは、ハミング距離の小さな値であるという仮定を基にした手法で、ハッシュ関数によって得られた値のハミング距離 1 以上の値から求められるバケットまで探索を行う手法である。これにより、既存手法に比べてメモリ空間の効率化と速度の向上といった全体的なトレードオフ関係における性能の底上げを行った。

この手法を用いた場合においても、LSH の概念を基本的な構造として用いているため、メモリ空間効率と精度、速度の間にトレードオフが解消されたわけではない。本論文では、この手法についても提案手法を適応し、提案手法の効果とその拡張性について評価、考察を行った。

2.4.2 Staged LSH のハードウェア実装

Staged LSH は、手法を FPGA を用いたハードウェア上にオフロードすることで手法の高速化を行っていた。FPGA は CPU と比較して、動作周波数が 1/10 ほ

どしかないため、ソフトウェア上で実行されるシーケンシャルな実装をそのままハードウェア上の演算器に対応付けたとしたとしても高速化を達成することができない。しかし、多くの演算器を持つ利点があるため、タスクを細分化し、タスク並列化を行うことで演算器の稼働率を上げ、高速化を実現可能である。Staged LSH では、大まかに要素の読み出し部と要素同士のハミング距離計算部にタスクを分割し、それぞれ独立して動作させることでタスクの並列化を実現している。

ここで、単に数ミリ秒以内の指紋の検索のみを考えた場合、ハッシュ関数におけるビット取得数 k に大きな値をとって、ハッシュ値の持つ値域を拡げることで要素を限りなく分割し、バケット内の要素を減らすことで、ソフトウェア上でも十分に高速化が可能となる。しかし、高い精度を維持することを考えると、多くのハッシュテーブルが要求される。メモリの増設に対して、物理的に、もしくはコスト的に糸目をつけなければ、ソフトウェア上で要求される高速化を達成できるが、現実的な制約があるため、大規模な指紋データベースを対象とするとき、ハッシュテーブルを格納するメモリ領域に限界が生じる。そこで、パラメータ k にある程度小さな値を取り、要求精度に求められるハッシュテーブル数を抑え、メモリに格納される要素の合計サイズを小さくした場合、バケット内に複数の要素が格納されることは避けられないため、タスク並列化による高速化に大きな利点が生まれる。

このように、メモリに格納する要素自体の数と合計サイズを小さく抑えたまま、一定時間内に処理可能な指紋データベースサイズ規模を拡張することができるため、FPGA デバイス上での検索処理の高速化は、メモリ空間効率と高速化のバランスが良い方法だと考えられる。また、ハードウェア上にハミング距離計算機などをうまく作りこむことで、高速に指紋同士の比較が可能であるアーキテクチャなども実現している。

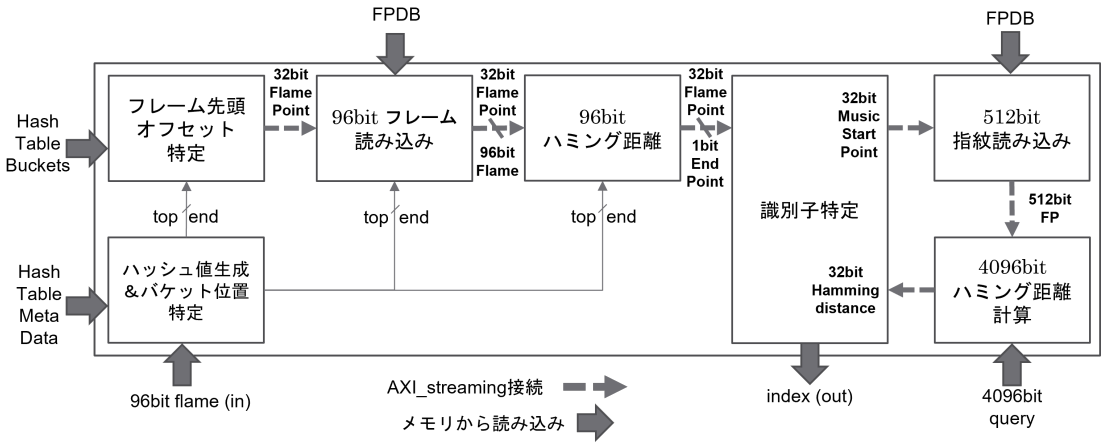

図 2.9: Staged LSH のハードウェア実装の構成

今回評価に用いた実装では、図 2.9 で表すようにタスクを細分化し、ハッシュ探索と 96 ビットハミング距離計算を行うカーネルと識別子特定および 4096 ビット

ハミング距離計算を行うカーネルで構成されている。ここで、図中の Hash Table Meta Data は、ハッシュテーブルの各バケット位置を指す番地を格納した、ハッシュ値の値域に対応したサイズの配列で、FPDB は指紋データベースである。これらの詳細なデータ構造については、2.5.1 項の図 2.11 に示す。

2.5 Staged LSH の分析

ここでは、既存手法である Staged LSH の FPGA デバイス実装における速度とメモリ空間効率、および並列化手法の分析と考察を行う。

2.5.1 FPGA プラットフォームとデータ構成

評価に使用するアクセラレータデバイスのプラットフォーム構造は図 2.10[19, p.19] のようになっている。デバイス内に同時に並列アクセス可能な 4 つのグローバルメモリ（DDR）があり、回路の実装部である各 SLR（Super Logic Region）ごとに小規模なメモリ（PLRAM）が実装されている。

FPGA 内部に実装されているメモリは、DRAM のように大規模なメモリ空間を確保することはできない一方、レイテンシが小さく高速にアクセス可能である [20, p.95]。そのため、データをこのようなカーネル内のメモリバンド幅が大きく、遅延の小さなローカルなメモリに格納することで、ハッシュテーブルを用いた探索の高速化を狙った研究として、文献 [21] のハッシュ関数により特定した対象のバケットをローカルのメモリへ読みだすことで、グローバルメモリへのランダムアクセス回数を減らしたオンチップバケット戦略や、文献 [18] の頻繁にアクセスするデータをローカルのメモリへ格納しておく戦略などがある。流行の楽曲などを考慮した手法は、実世界の著作権管理システム上で有効であるが、今回は大規模な楽曲データに対して、純粋な探索速度の評価を行い提案手法の効果を検証するため、すべての要素をグローバルメモリへ格納した際の評価を行った。

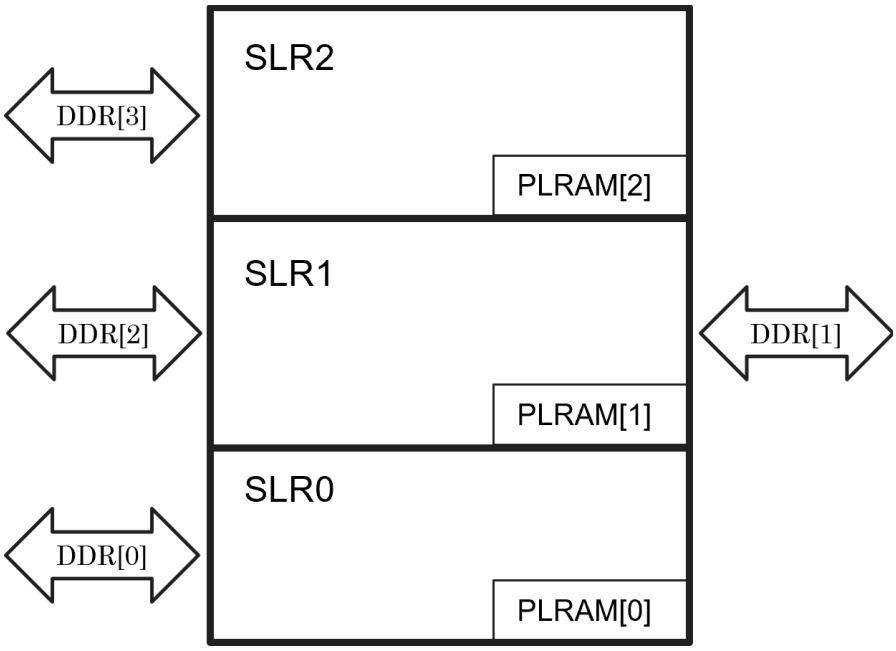

図 2.10: AlveoU200 プラットフォーム

次に、データを格納しているデータ構造について議論する。LSH を用いた手法では、要素の探索のために指紋データベースとハッシュテーブルの 2 つのデータが必要となる。ここで、ハッシュテーブルを構築するためのデータ構造には以下のような手法とそれぞれの手法の特性がある。

- **リスト構造を用いたハッシュテーブル**

  データの追加や削除が容易で、データ構造の構築自体にも複雑な処理が不必要であり、直感的であるが、データを格納しているデータ部に加えて、アドレス部が必要なため、無駄なメモリ空間を必要とする欠点がある。

- **配列構造を用いたハッシュテーブル**

  データの追加や削除のたびにデータ構造の大きな書き換えを必要とするが、アドレス部の必要がないため、メモリ空間の無駄が小さい利点がある。

また、ソフトウェア上での実装では、リンクリストとベクターのハイブリッドなデータ構造を利用したハッシュテーブルを用いることで、検索性能とメモリ使用量の効率化を行う手法なども提案されている [22]。

今回のようなハードウェアにおける実装を前提とする場合において、面積や単価によるメモリの制約がより重視される可能性が高いことを念頭に置くと、無駄

なメモリ空間を圧迫する要素は、追加や削除の手間に比べて優先して排除すべき点であると考えられるので、今回の評価においては、図 2.11 のような構造を取る、配列を用いたデータ構造を構築する。

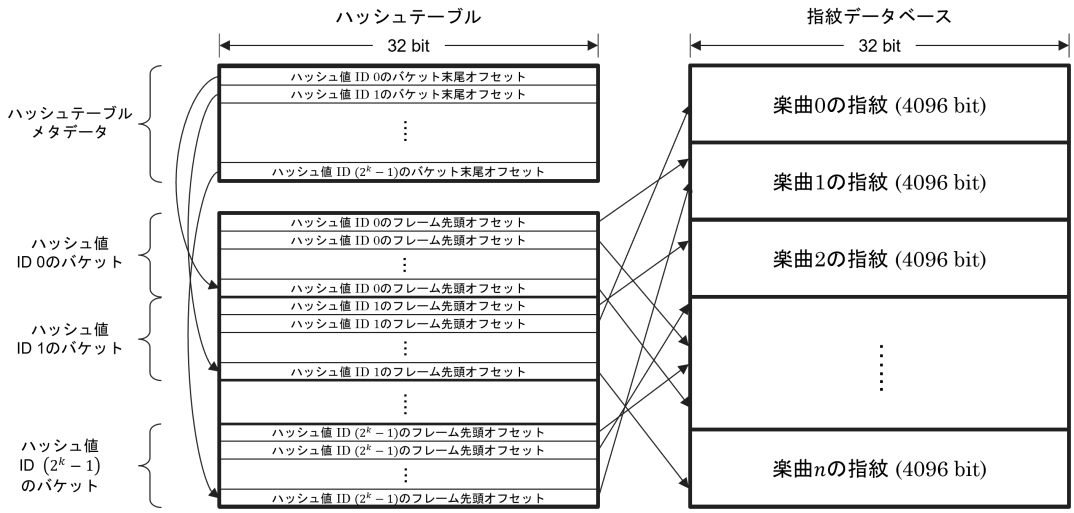

図 2.11: Staged LSH で用いるデータ構造

指紋データベースには、各楽曲から生成した n 曲分の指紋を 32 ビットのサブ指紋をひと固まりとして格納し、ハッシュテーブルには、バケット部にデータベース上の各フレームの先頭位置を指すオフセットを格納する。また、メタデータ部には、各ハッシュ値に対応するバケットの末尾位置を格納している。

以上のデータ構造上でハッシュ値を h、ハッシュテーブルメタデータへのアクセスを関数 M とすると、バケットの先頭位置 top とバケットの末尾位置 end は式 (2.2)、式 (2.3) で求められる。これらを用いてバケットのサイズを求め探索範囲を決定する。

$$\text{top} = \begin{cases} 0, & if\ h = 0 \\ M(h-1), & if\ M(h-1) = 0 \\ M(h-1)+1, & if\ M(h-1) \neq 0 \end{cases} \tag{2.2}$$

$$\text{end} = \text{M}(h) \tag{2.3}$$

2.5.2 実装環境

表 2.1: ソフトウェア実装環境

Host	alveo20
CPU	AMD Ryzen 7 3700X 8core 16thread 3.6GHz
Host Memory	DDR4 3200 MT/s 128GB
OS	Ubuntu 18.04
Compiler	GNU Compiler Collection version 7.5.0
Compiler Option	-std=c++11, -O 3, -mcmodel=medium
Library (Time)	std::chorono::system_clock

表 2.2: ハードウェア実装環境

Host	alveo20
CPU	AMD Ryzen 7 3700X 8core 16thread 3.6GHz
Host Memory	DDR4 3200 MT/s 128GB
OS	Ubuntu 18.04
FPGA	Xilinx Virtex UltraScale + XCU200
Platform	U200 XDMA 201830_2
Global Memory	DDR4 16GB × 4 64GB
PLRAM	128KB
Compiler	Vitis Compiler version 2020.2
Vitis Compiler Option	-O 3
Library (Time)	clGetEventProfilingInfo (CL_PROFILING_COMMAND_START, CL_PROFILING_COMMAND_END)
Library (Memory Size)	malloc_usable_size

実装した回路は、並列実装時に要求されるリソースサイズの都合上、図 2.10 のプラットフォームにおける SLR0 上に合成している。FPGA 上のカーネルの時間の測定においては、FPGA デバイスへの各要素の転送時間および、DMA のセッ

トアップの時間は含まず、clGetEventProfilingInfo を用いて、デバイスでカーネルの実行を行うコマンドをキューに入力する API である clEnqueueTask の実行がデバイス上で開始したときのデバイスタイムカウンタの値と終了時の値の差から時間を求め、500 回試行した場合の平均時間を算出している。また、並列実装の評価では、各カーネルの開始時刻誤差を含む。実験に使用する指紋は大規模なデータベースを想定するため、乱数生成器 std::random_device を用いて生成したものを用いた。評価に使用するアクセラレータカードを図 2.12 に示す。

図 2.12: 評価に用いた Alveo U200 データセンターアクセラレータカード

2.5.3 Staged LSH のソフトウェア実装との比較

表 2.3: 既存手法の評価に用いたパラメータ

パラメータ	設定値
ハッシュ関数 ビット取得数 k bit	13
ハッシュテーブル数 l bit	4
スクリーニング閾値	24
精査閾値	1024

評価に用いたパラメータについては 3.3 節で決定するものを用いている。

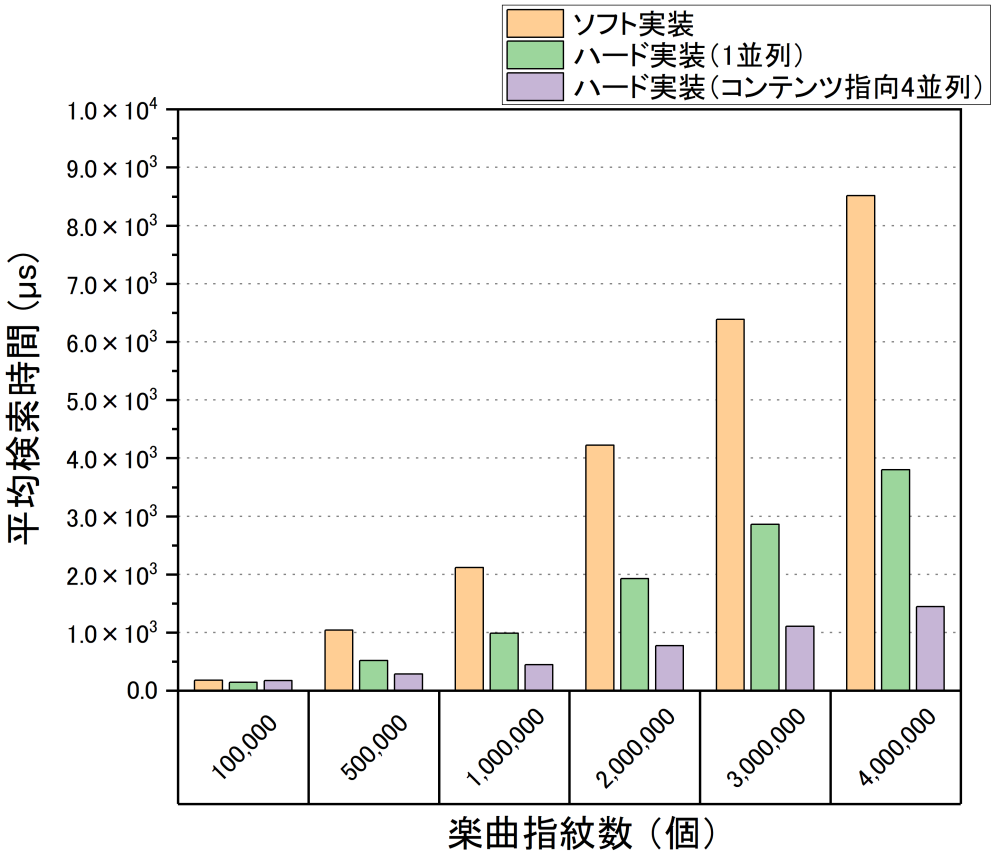

図 2.13: ハードウェア実装とソフトウェア実装の速度比較 ($BER = 0.00$)

Staged LSH をソフトウェア上とハードウェア上で実装し、指紋データベースサ

イズを変えながら 500 回の探索時間の平均値を測定した。結果を見ると図 2.13 のように大規模な楽曲指紋データベースにおいて特にハードウェアにおける実装の利点が大きいことが分かる。また、4.2 節で考案する適切な回路の並列化を行うことで、既存手法においてもハードウェア上で処理の高速化が容易であることも確かめた。この結果から、面積や単価面でよりメモリ容量に対して制約の大きいハードウェアにおける省メモリ化の要求が大きい時、また、速度の制限を維持したままメモリ効率をよくしたい場合、本研究の目的である省メモリ化および高速化の意義が大きいことを再確認した。

2.5.4 ボトルネックと課題の分析

文献 [17] では、各データの格納に FPGA 内部の SRAM を用いることで、メモリへのアクセス速度の影響を限りなく小さく抑えられていたが、SRAM の容量の都合上、数百楽曲分の指紋までしかデータを格納することができなかった。しかし、実際に著作権管理システムにおいて用いられる楽曲は、非常に大きなデータベースを対象とするため、各要素を格納する保存領域の最有力候補としては、大きな容量を確保することができるグローバルメモリとなる。一方、回路により近い位置にある SRAM と比べてレイテンシが非常に長くなり、検索時間におけるメモリへのアクセス速度の影響が大きくなることが懸念される。

中でも、スクリーニングを行うカーネルにおける指紋データベースからのフレームの呼び出しは、ランダムアクセスを大量に必要とする。そのため、表 2.4 のようにスクリーニングカーネルの実行時間は、外部メモリへのアクセスの待ち時間が全体の 75%以上と大部分を占めることが分かった。また、1 並列の実装における検索時間は、指紋データベースからのフレームの転送回数にほとんど比例して増加することを確認した。

表 2.4: Staged LSH のメモリアクセス解析 (指紋数:100 万, 試行回数:100, $BER = 0.50$)

スクリーニング CU	並列数	外部メモリアクセス Stall 時間が CU 実行時間に占める平均割合 [%]	平均フレーム転送速度 [MB/s]	平均フレーム転送回数 [回]	最悪 CU 呼び出し回数 [回]
Staged LSH	1	78.807	405.614	1,744,245,900	50,400
Staged LSH（コンテンツ指向並列版）	4	76.237	305.327	436,099,111	50,400

また、LSH を用いた手法では精度の面でルックアップテーブルを用いた手法と比較して有利になった一方、精度とメモリ空間効率および検索速度の間にはトレードオフが存在する。以下でそのトレードオフにおける、実装上の課題について確認する。

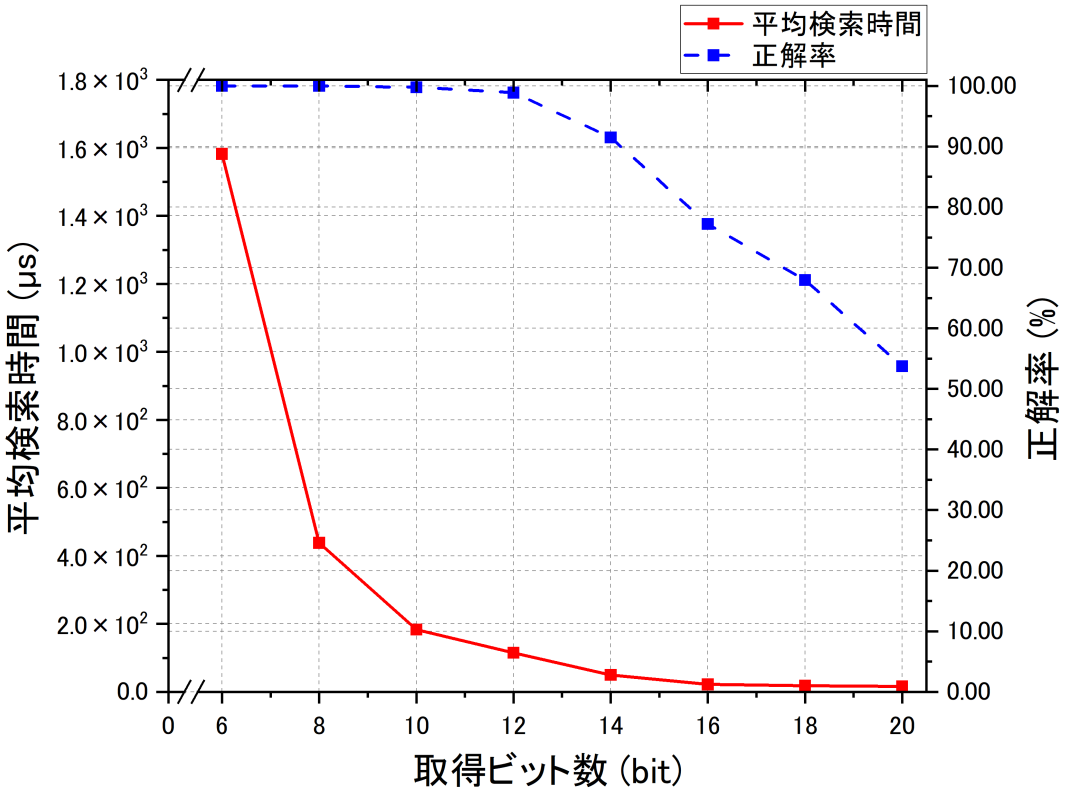

図 2.14: ハッシュ関数の持つ値域による精度と速度の推移 (指紋数:1 万, 試行回数:1 万,$BER = 0.25$,$l = 1$)

図 2.7 で表された任意位置のビット取得で構成されるハッシュ関数において、図 2.14 のように取得ビット数が増加する、つまり、ハッシュ値の持つ値域が拡大するほど、ハッシュテーブルにおけるバケット数の増加により、データを細分化可能であるため、検索に掛かる時間を短縮することが可能であることが分かる。しかし、同一楽曲から生成された歪みを含んだ指紋に対して、同一のハッシュ値が得られる確率が小さくなるため、指紋の発見確率は減少する。このような速度と精度のトレードオフに対して、ハッシュテーブルを複数用意することで、バケット内の要素の探索速度の向上を図ったまま、一定以上精度を確保する方法が用いられる。

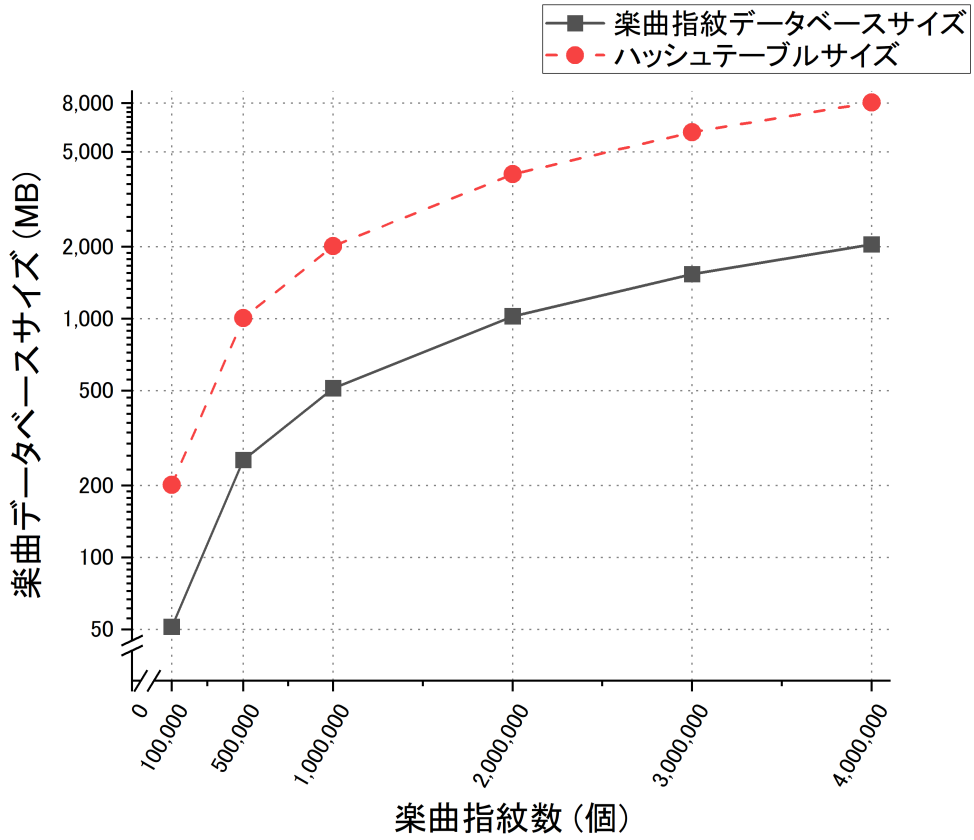

図 2.15: $BER = 0.25$ のとき 99.99%以上の正解率保証に必要な各要素サイズの推移

今回評価のために精度の基準として使用した、BER0.25 の時、99.99%以上の検索精度[*1]を保証するには、3.3 節より、4 つのハッシュ関数とそれぞれに対応するハッシュテーブルが必要となる。しかし、図 2.15 のようにデータベースと比較して、ハッシュテーブルの合計サイズは約 4 倍にもなることが分かる。これらから、大規模な楽曲データベースを想定する場合、ハッシュテーブルのメモリ容量の圧迫が課題となることが容易に想像できる。2.4.1.1 目において関連研究として挙げた隣接 Staged LSH においても、精度を向上させる場合には、ハッシュ関数を複数用いてメモリ空間効率を犠牲にするか、あるいは、フレームから求めたハッシュ値よりハミング距離が 1 より大きな値から特定されるエントリを探索することが必要となる。

[*1] ここでベンチマーク時に用いた精度については 5.2.1 項で考察する。

2.5.5 既存並列化戦略の分析

FPGA デバイス上に回路を実装する上で処理の効率向上を目的とし、処理回路を複数並列に実装することは処理の高速化を達成するための単純な戦略と言える。Staged LSH の文献内では、複数のハッシュテーブル間に依存性がないことを利用した、図 2.16 のようなハッシュ関数方向に処理を分割した並列化が提案されている。以降では、この手法をハッシュ関数指向の並列化と呼称する。

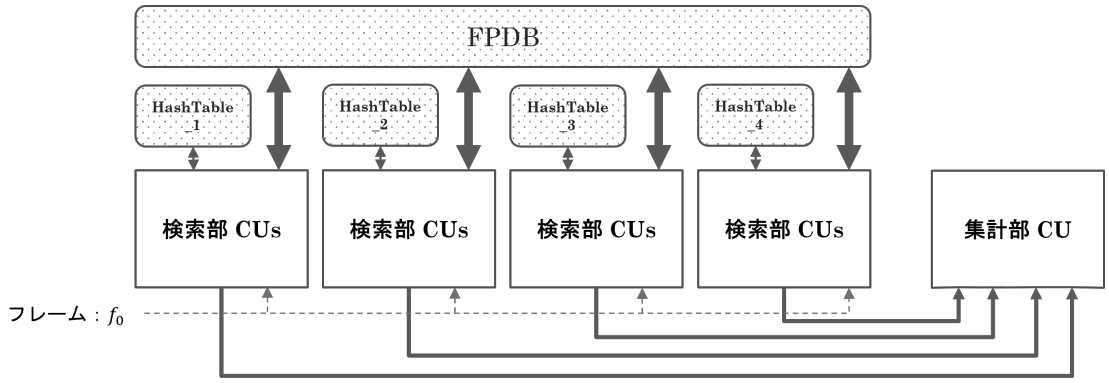

図 2.16: ハッシュ関数指向並列化における回路とデータ構造

個々の検索部は 2.4.2 項の図 2.9 で示した構造を持ち、それぞれの検索部に異なるハッシュ関数として機能する回路が構成されている。これによって、すべての検索部に同時に同じフレームを入力することで、一度にすべてのハッシュテーブルを探索することを可能としている。集計部ではすべての結果を集計し、もっともビットエラー数の小さい近似した結果を最終的な出力とする。

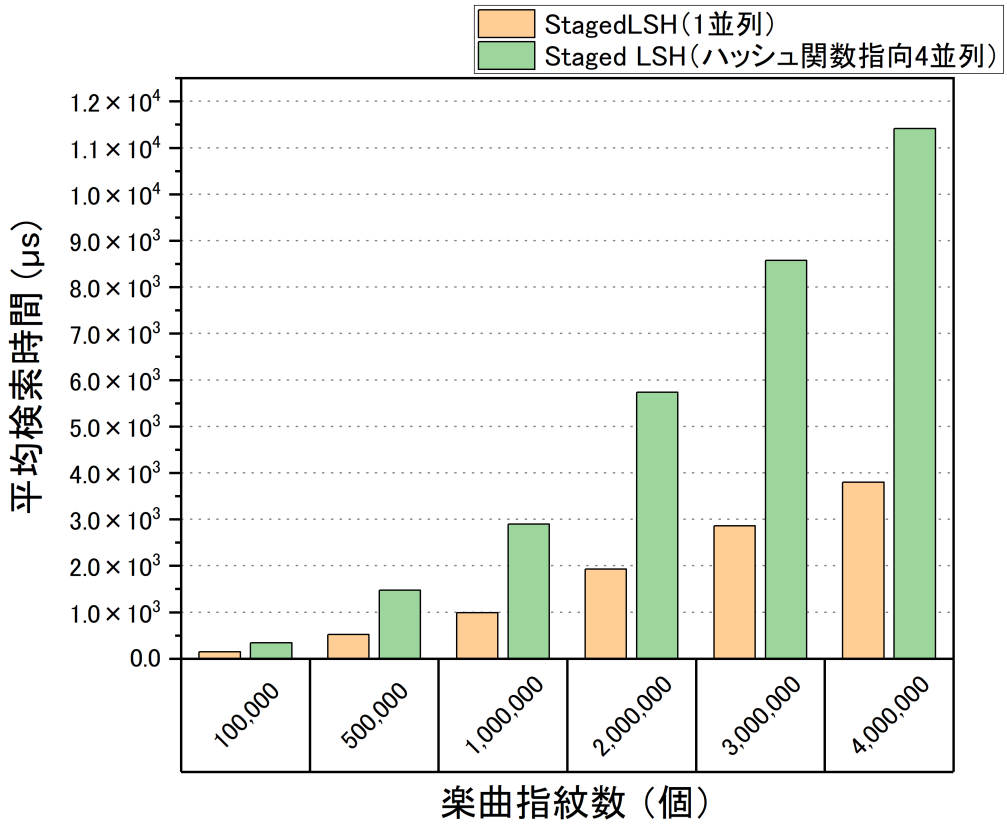

図 2.17: ハッシュ関数指向並列化の速度評価 ($BER = 0.00$)

表 2.5: ハッシュ関数指向並列化のメモリアクセス解析 (指紋数:100 万, 試行回数:100, $BER = 0.50$)

スクリーニング CU	並列数	平均フレーム転送速度 [MB/s]	平均フレーム転送遅延時間 [ns]
Staged LSH	1	405.614	232.298
Staged LSH（ハッシュ関数指向 4 並列版）	4	210.693	871.945

ハッシュ関数指向の並列化では、今回のように外部メモリであるグローバルメモリを要素を格納する主要なメモリとして用いる場合、単一のデータ構造である指紋データベースに対する複数の検索部からの読み出しアクセスは競合している。このような場合、同じグローバルメモリへのアクセスはアービタ回路によってシーケンシャルに処理される [23, p.124]。そのため、表 2.5 から分かるように、指紋データベースへのメモリアクセス速度がボトルネックとなる。また、500 回の検索

の平均検索時間を示した図 2.17 の結果からも分かるように類似度の高い指紋の検索では、1 並列のものと比較しても速度が低下することが分かった。

さらに、並列数においても回路を展開可能な上限数はハッシュテーブル数に従うという制約があり、今回のように、総ハッシュテーブルサイズを可能な限り削減し、省メモリ化を目的の主軸とするような場合においては、ハッシュ関数指向の並列化手法は、目的の方向性と相性が悪い場合がある。そこで以上のような状況における、新しい並列化のための手法を考案する必要がある。

2.6 おわりに

本章では、電子指紋とその概要についてまとめた。次に、本論文で対象となる指紋の生成手法について述べた。また、既存の検索手法である Staged LSH について分析を行い、グローバルメモリを用いた実装において検索速度がメモリからの要素読み出し回数に強く依存することが分かった。さらに、一定以上の精度を保証するために、指紋データベースと比較しても非常に大きなハッシュテーブルを格納するメモリ空間が必要であることが分かった。最後に、ハッシュテーブルの依存性がないことを利用したハッシュ関数指向の並列化戦略では、指紋データベースへのアクセス速度低下による検索部のスループット低下により、類似度の高い指紋の検索においては高速化できないことを確かめた。

第3章　楽曲電子指紋の時間的連続性に着目した検索手法

3.1　はじめに

2.5 節における手法の分析から、LSH を用いた手法で高い精度で最近傍要素の検索を行うには、データベースと比較しても非常に大きなメモリ空間を占める複数のハッシュテーブルが要求される場合があり、このことがハードウェア上に手法を実装する上で致命的な欠点にもなることが分かった。また、グローバルメモリへのランダムアクセスの数を減らすことができれば、根本的な検索の高速化が可能であることも明らかになった。そこで、LSH を用いた”楽曲”電子指紋の検索において、指紋の時間軸の連続性に着目した改良を行う。

本章では、LSH を用いた手法のトレードオフである、検索速度、精度、メモリ空間効率の性能を底上げする手法について考察し、評価のためのパラメータの決定についても議論する。また、ルックアップテーブルを用いた手法と提案手法の立ち位置について議論する。

3.2　省メモリ化および高速化戦略

2.3 節の電子指紋の生成手法を見ると、指紋を生成する過程で楽曲の時間軸に対して特に操作は行っておらず、生成された指紋と元の楽曲の時間軸に対する同一性は失われていない。また、ハミング距離により類似度の比較を行うことを前提とした手法では、指紋に時間的なズレがないことは明らかである。そこで、各フレームから生成される既存のビットサンプリングで構成されるハッシュ値に対して、フレームの指紋先頭からのオフセット情報、つまり時間の付加情報を与えることができる。

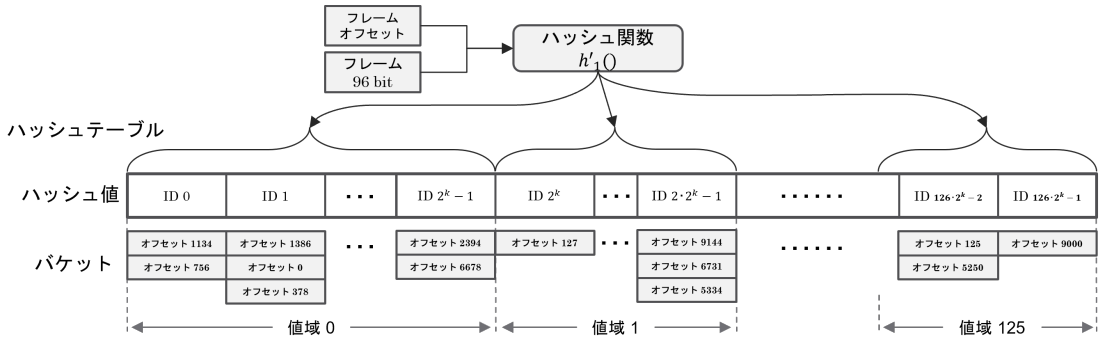

図 3.1: 時間的連続性に着目した手法による検索と格納方法

図 3.1 のようにハッシュ関数に対して、96 ビットのフレームと同時にフレームの時間軸に対する情報であるフレームの先頭からのオフセットを入力として与える。与えられたオフセット情報をもとに、クエリ指紋から取り出した各フレームに対して、対応するそれぞれ独立した値域を割り当てることで、ハッシュテーブルの持つバケットの総数を大幅に増加させることができる。これにより、入力に対してハッシュ値の衝突する確率を大幅に減少させることが可能であるため、必然的に各バケット内の格納要素数を減らすことができ、グローバルメモリへのランダムアクセス回数を削減することができる。

既存の手法であれば、値域の拡大には検索精度、あるいは、メモリ空間効率のいずれかを犠牲にするが、本手法では値域の拡大に、そのどちらも関与しないため、速度とメモリ空間効率の底上げが可能となる。

また、ハッシュテーブル内の要素の格納について、要素の性質上、あるいは、ハッシュ関数の性質上、偏った配置となることが懸念される。偏った要素の分布は、検索時間に影響を与えるため、値域サイズの拡大に伴い、検索速度を効率よく向上させるには全てのバケットに要素が均等に格納されることが理想的である。ここで、提案手法では、新たに時間軸による分割を行うため、各バケットの要素数の偏差を既存手法に比べて小さくする効果も期待できる。

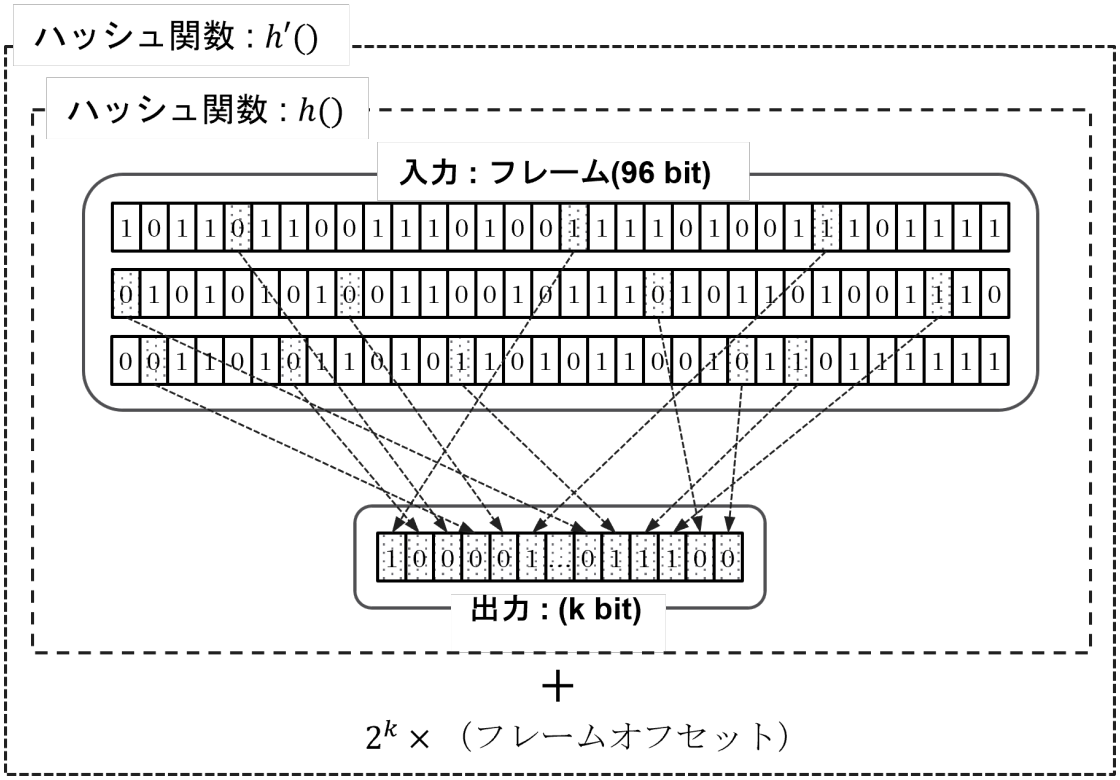

図 3.2: 時間的連続性に着目した手法で用いるハッシュ関数

図 3.1 のようなデータ構造に対応するハッシュ値の生成のため、本手法で用いるハッシュ関数 h' の具体的な構成としては、図 3.2 のようになっており、指紋から取り出された各フレームに対して、独立した値域を対応付ける処理を行うことになる。これによって、各フレームが指紋生成に用いた楽曲のサンプル先頭からの位置に応じたバケットへ割り当てられる。本手法で着目しているのは、楽曲から生成された指紋の構造そのものであるため、様々なハッシュ関数 h に対して拡張の柔軟性があるはずである。そこで、2.4.1.1 目で紹介した隣接 Staged LSH で用いられるハッシュ関数へも本手法の適応が可能であると考えられる。

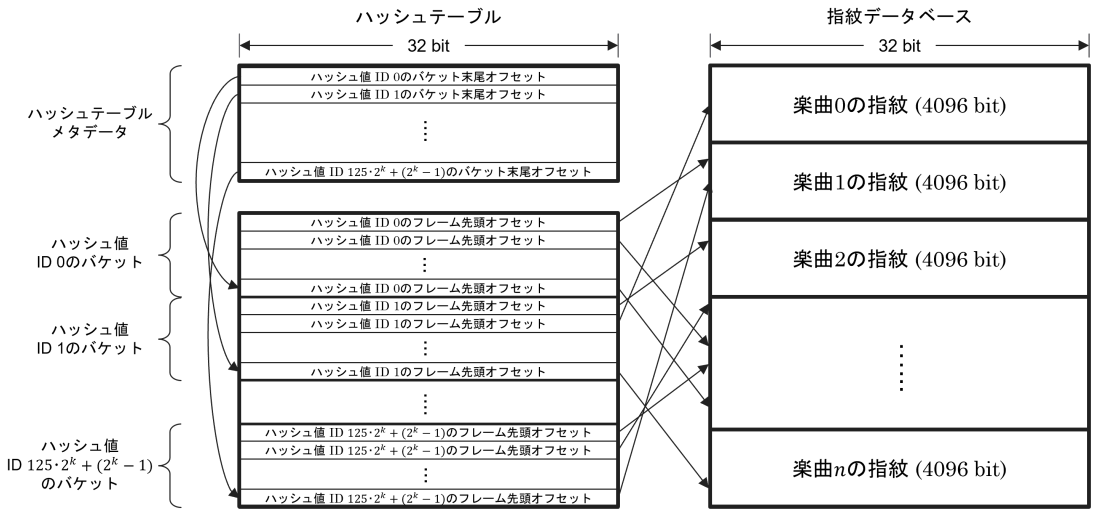

図 3.3: 時間的連続性に着目した手法で構成されるデータ構造

上記手法を反映した各データの構成は図 3.3 のようになる。本手法により、既存の Staged LSH や隣接 Staged LSH において、検索精度を同等に維持したまま、メモリ空間効率を同等、あるいは向上させ、検索速度についても向上させることが可能であると考えた。以降、図や表中では提案手法を時間連続構造と略記する。

3.3 評価用パラメータの決定

既存の LSH を用いた検索手法と、今回の提案手法を採用した検索手法の精度を一定の精度以上で同等とし、手法の優位性を評価するためのパラメータを検討する。

まず、ビットエラー率が BER の時の、1 フレームあたりに含まれる平均ビットエラー数の近似値 r は式 (3.1) のように求められる。

$$r(BER) \fallingdotseq 96 \cdot BER \tag{3.1}$$

また、データベース内に対応する指紋が存在する楽曲が入力されたとき、生成された指紋から取得される 126 フレームすべてを処理しても、正解の楽曲と特定することができない確率 P_f は、ハッシュ関数の数を l、ハッシュ値を構成するビット数を k とすると、式 (3.1) を用いて式 (3.2) のように近似値を求められる。ここでは、まったく異なるフレームから正解の楽曲が偶然特定されるような状況は想定していない。

$$\mathrm{P_f}(BER) \fallingdotseq \left[\left\{ 1 - \left(\frac{96 - r(BER)}{96} \right)^k \right\}^l \right]^{126} \tag{3.2}$$

今回は、パラメータの決定には、$BER = 0.25$ の指紋データベース内に対応する指紋の存在する指紋が入力した時、正解率が 99.99% 以上となることを同等な精度の基準として用いる[*1]。この精度を満たす手法評価のためのパラメータ k と l を求める。システムに用いるハッシュ関数の数 l の値をできる限り小さくすることで、指紋データベース内に存在しない指紋の入力に対して、処理の大幅な低速化を回避することができる。そのため、k の値を小さくして、一定の精度制約に対して、l の値の増加を抑える利点が大きいが、数百万曲分以上の指紋を格納したデータベースに対しては、ある程度のバケットの分割を行わなければ、バケット内の検索時間が膨大となる。そこで、文献 [17] で用いられていた $k = 13$ を既存の Staged LSH で評価の基準とし、精度要件を満たす最小の合計ハッシュテーブルサイズとなるハッシュ関数の数 l を決める。また、最小の合計ハッシュテーブルサイズをとり、既存のハッシュテーブルの値域以上の最大の値域幅を持つ提案手法のパラメータ k を決定する。

ハッシュテーブルでのバケット分割数は、既存手法の Staged LSH と、ハッシュテーブルに指紋連続構造を持たせた指紋の持つ時間的連続性に着目した手法を適用したものとで、ハッシュ関数内で選ばれるビット数に関連するパラメータ k の値によって表 3.1 のように遷移する。提案手法では、既存手法と比較して、同じハッシュ関数のビット取得数であっても 126 倍の値域幅を持つことが分かる。

表 3.1: ハッシュ関数の取得ビットと値域

ハッシュ関数 ビット取得数 k bit	Staged LSH 値域幅	Staged LSH+指紋連続構造 値域幅
7	128	16,128
8	256	32,256
9	512	64,512
10	1,024	129,024
11	2,048	258,048
12	4,096	516,096
13	8,192	1,032,192
14	16,384	2,064,384

本来は、デバイスの持つメモリ条件やシステムの時間的制約によって、パラメータを選ぶ際の条件は様々であるが、今回パラメータを決定する際の条件の優先順位は、速度に対して省メモリ化を優先事項として以下に示すようにした。

[*1] ここでベンチマーク時に用いた精度については 5.2.1 項で考察する。

条件 (1) $BER = 0.25$ の指紋の入力に対して、正解率が 99.99%以上であること

条件 (2) 条件 (1) を満たし、最小限の数のハッシュテーブルを取る、パラメータ l の最小値を選択すること

条件 (3) 条件 (1) を満たし、ハッシュ関数が最大の値域幅を取る、パラメータ k の最大値を選択すること

また、既存の手法と提案手法を適用したものとをソフトウェアで実装し、パラメータを決める判断のための材料の一部とした。10 万個の指紋を格納したデータベースについて、クエリ指紋に対してランダムに $BER = 0.25$ のビットエラーを付与した指紋を入力として、実際に 10 万回の試行を行った結果、提案手法では表 3.2、既存手法では表 3.3 のようになった。それぞれ、k もしくは l のどちらかのパラメータを固定したときの、もう一方のパラメータの変動による精度の遷移を表している。

表 3.2: 提案手法のハッシュ関数における取得ビット数と正解率（$l = 1$）

ハッシュ関数 ビット取得数 k bit	正解率 [%]
7	100.000
8	99.994
9	99.994
10	99.940

表 3.3: 既存手法における用いるハッシュ関数の数と正解率（$k = 13$）

ハッシュ関数の数 l 個	正解率 [%]
1	96.522
2	99.755
3	99.983
4	99.999

上記式 (3.2) とソフトウェアでの実測値の結果の両方を踏まえ、提案手法の評価のための同等精度のパラメータとして既存手法のパラメータを $k = 13$、$l = 4$ とした。また、指紋の持つ時間的連続性に着目した手法では、取得ビット数のパラメータ k が 8 と 9 の時、精度が完全に同じとなったため、ハッシュ関数内部のビット取得位置や、クエリへのビットエラーの分布の差による精度の揺らぎに対して、精度の条件値との余地を大きく取ることを考慮し、今回はパラメータを $k = 8$、$l = 1$ とした。また、システムの検索精度については今回ベンチマーク用に選択したもの以外にもパラメータによって柔軟に設定可能である。

3.3.1　メモリ効率と速度の期待値

ここで選んだパラメータから、時間的連続性に着目した手法ではハッシュ関数おいて既存の Staged LSH と比較して約 4 倍の値域幅を取ることが可能であるため、約 4 倍程度の高速化が期待できる。また、隣接 Staged LSH と比較すると、パラメータを揃え、検索精度を同等としたうえで、ハッシュ関数が約 126 倍の値域幅を取ることができるため、十分に要素で満たされたハッシュテーブルにおいて約 126 倍の高速化が期待される。

ハッシュテーブルのメモリ使用量については、既存手法と提案手法を適用したものとで推定要求サイズを以下のように計算可能である。各配列へのオフセット（番地）を保存するサイズを a バイト、楽曲指紋数を n すると、既存の Staged LSH では式 (3.3)、ハッシュテーブルに指紋連続構造を与えたものでは式 (3.4) となる。

$$\left\{\left(2^k \cdot a\right) + (126 \cdot a \cdot n)\right\} \cdot l \tag{3.3}$$

$$\left\{(126 \cdot 2^k \cdot a) + (126 \cdot a \cdot n)\right\} \cdot l \tag{3.4}$$

同等の精度を達成するために双方に同じパラメータを選択した場合、式 (3.4) のほうがわずかに、理論上メモリ使用量が大きくなる。しかし、時間的連続性に着目した提案手法では、既存の手法と比較して、同等以上の精度および速度を達成するために、パラメータ k および l により小さな値を取ることができる。

ここで、十分に大きな指紋データベースサイズを仮定すると、今回選んだ、精度を一定以上で同等とするパラメータにおいて、式 (3.3) では $l = 4$、式 (3.4) では $l = 1$ を取るため、既存の Staged LSH 手法と比較して提案手法を適応したものでは、ハッシュテーブルのメモリ使用量は約 75%程度の削減が期待できる。

3.4　ルックアップテーブルを用いた手法への検討

電子指紋の探索手法として、文献 [12] で提案されていた方法では、指紋を 1 つあたり 32 ビットのサブ指紋と呼ばれる要素に分割し、それぞれのサブ指紋をルックアップテーブルを参照するキーとして用いていた。これによって、それぞれのサブ指紋の指紋データベース上に対応する位置を 2^{32} 個のルックアップテーブルのバケットに分割して格納し、探索範囲の削減を行うことを可能とし、大規模なデータベースに対して単純な検索手法と比較して高速化を達成した。

探索範囲の削減を行う目の付け所としては、時間的連続性に着目した今回の提案手法の高速化戦略と同じと言える。しかし、ルックアップテーブルのエントリ数に柔軟性がなく、エントリ数が対象の楽曲データ規模に比べて大きすぎる問題がある。表 3.4 に示すように、1 曲あたり 4096 ビットの指紋を用いる場合、ルック

アップテーブルの各バケット内には、平均的に 1 つ以上の要素が格納されずルックアップテーブルが十分に利用できていないことが分かる。

表 3.4: 2^{32} のエントリを持つルックアップテーブルのバケット当たりの平均格納数理論値

楽曲指紋数 [個]	サブ指紋数 [個]	バケット当たりの平均格納数 [個]
100,000	12,800,000	0.003
1,000,000	128,000,000	0.030
2,000,000	256,000,000	0.060
3,000,000	384,000,000	0.089
4,000,000	512,000,000	0.119
5,000,000	640,000,000	0.149

しかし、今回 3.3 節で選んだ $k = 8$ のパラメータで、指紋の時間的連続性に着目した手法を用いると表 3.5 のように、各バケットに対して十分に要素が満たされ、ハッシュテーブルが無駄なく利用されていることが分かる。また、パラメータ k の値を調節することで、柔軟に探索範囲と精度を変えることができる点も、LSH を用いる都合の良い点だと言える。速度の点だけ考えると、ルックアップテーブルを用いた手法は、高速な検索を可能とするので多少のメモリ空間効率の無駄がある程度で、高速化に有効な手法である。しかし、指紋を用いた指紋の検索精度について以下のような課題がある。

ルックアップテーブルを使用した手法は、異なる手法により圧縮された同一楽曲から生成される指紋には、少なくとも 1 つビットのエラーの無いサブ指紋が存在することを前提としている。そのため、ビットエラー率が 0 に限りなく近く、同一楽曲から生成された指紋同士の類似度が比較的高いことが要求される。しかし、楽曲の歪みが大きく、生成される指紋の類似度の低い場合には、サブ指紋当たりにビットエラーの含まれる確率が高いため、ルックアップテーブルを用いた手法では十分な検索精度が保証できない。そのため、ビットの信頼度等を用いることで精度を上げるなどの工夫が必要となっている。ハミング検索など複数の解候補を選出するような場合には、ある程度の検索精度でも構わない場合があるが、著作権管理システムのような、高い精度が求められる場合には、精度に柔軟性のある LSH を用いた手法が望ましいと考えられる。

表 3.5: $2^8 \times 126$ のエントリを持つハッシュテーブルのバケット当たりの平均格納数理論値

楽曲指紋数 [個]	フレーム数 [個]	バケット当たりの平均格納数 [個]
100,000	12,600,000	390.625
1,000,000	126,000,000	3906.250
2,000,000	252,000,000	7812.500
3,000,000	378,000,000	11718.750
4,000,000	504,000,000	15625.000
5,000,000	630,000,000	19531.250

メモリ効率の観点から見ると、ルックアップテーブルを用いた手法では、すべてのサブ指紋位置に対して、データベースに対応する位置を保存する。そのため、ルックアップテーブルへの最大エントリ数は、フレームの先頭を格納する場合と比べてわずかに大きくなる。また、楽曲から生成された指紋では、サブ指紋から確定するエントリ内の要素数の分布に最大 20 倍程度の偏りが生じることが分かっている。

ルックアップテーブルを用いた手法へ、今回の指紋の持つ時間的連続性に着目したデータ構造を適応すること自体は可能である。しかし、十分な精度を保証できるが、速度やメモリ効率の点で劣る Staged LSH に対して、今回の提案手法を用いて、性能の底上げを行うことで、指紋を用いたシステムの汎用性の向上が見込めるため、今回は Staged LSH を対象とした。

3.5 おわりに

本章では、LSH を用いた電子指紋の検索において、楽曲から生成される指紋の時間的連続性に着目し、データ構造および、格納、検索手法に対して改良を加え、性能の底上げを行う手法を考案した。また、手法において用いる各種パラメータの決定方法について議論し、提案手法の評価に用いる既存の手法と同等の精度達成する各種パラメータを決定した。最後に、既存のルックアップテーブルを用いた手法とハッシュテーブルを用いる手法について比較し、提案手法の立ち位置と役割について明確にした。

第4章　ハードウェアにおける指紋検索の並列化

4.1　はじめに

FPGA 内部の、回路により近い場所に実装されているローカルなメモリは、グローバルメモリのように大規模なメモリ空間を確保することができない一方、高速にアクセス可能であるためボトルネックとなっていたフレームの読み出し処理に対する影響を抑制できる。しかし、ハッシュテーブルや指紋データベースといった要素を外部メモリであるグローバルメモリを中心として格納する場合、指紋データベースからのフレームの読み出しがボトルネックとなっていた。また、分析から明らかなように、2.5.5 項で議論したハッシュ関数指向の並列化では、並列に実行される複数の回路からの指紋データベースへの読み出しリクエストがシーケンシャルに処理されるため、メモリアクセスに対するストール時間がより増加し、高速化の足を引っ張っていた。加えて、ハッシュテーブル間に依存性が無いことに着目した手法は、3.3 節で決定した、時間的連続性に着目した手法の評価のためのパラメータのように、省メモリ化を意識して、ハッシュテーブルを意図的に減らした実装を考慮する必要がある。つまり、今回のような省メモリ化のコンセプトに適合していない並列化手法だと言える。そこで、新しい並列化手法を考案する必要がある。

本章では、上記の課題と指紋検索における特性に着目した 2 種の並列化戦略について議論する。

4.2　コンテンツ指向並列化

文献 [22] では、クラスタ環境下で複数のデータノードに対して、指紋データベースおよび、ハッシュテーブルを各クラスタに分割して配布することによって、負荷分散による高速化を図っている。コンテンツの分散により、各データノードがローカルに類似度の計算を行うことによって、検索性能の向上を達成した。また、システムで頻繁に検索される楽曲や、バックアップ用のノードなど特別なデータノードを設けることでパフォーマンスとサービスの可用性を向上させた。この概念を FPGA 上の並列化に適応することで、ハッシュ関数指向の並列化における課題の解決を目指す。

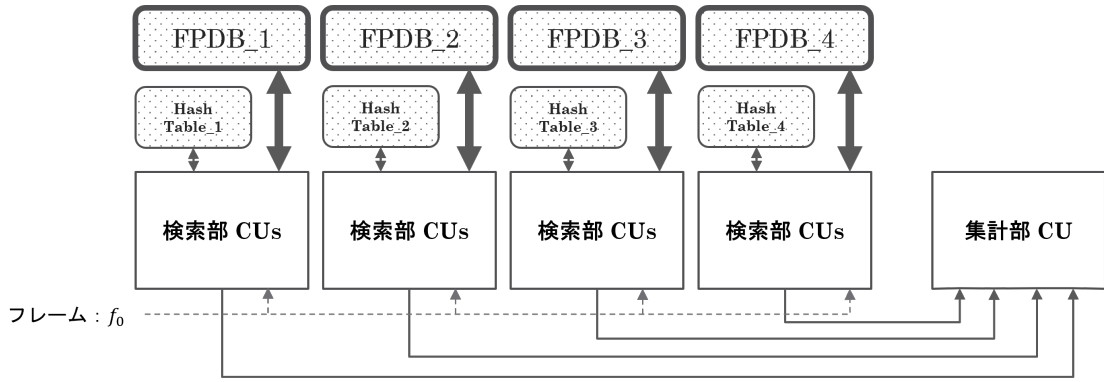

図 4.1: コンテンツ指向並列化における回路とデータ構造

図 4.1 のように、それぞれの検索部に一対一に対応する指紋データベースが存在するように要素を分割し、それぞれに対応したハッシュテーブルを構成する。そうすることで、既存のハッシュ関数指向の並列化で課題となっていた指紋データベースへのアクセス競合による検索時間への影響の低減が期待できる。また、データベースを分割することで、各指紋データベースの要素数が 32 ビットの符号なし int 型で表現できるサイズを超えることを抑制できる利点がある。これによって、対応するハッシュテーブル内に格納している各フレームを指すオフセットを表現する値のデータ型を大きくすることなく、ハッシュテーブルの無駄な容量の増大を抑えることができる。

今回の実装では、2.5.1 項の図 2.10 で示した、プラットフォーム上に接続されている並列アクセス可能な 4 つのグローバルメモリに対して、指紋データベースとハッシュテーブルをできるかぎり分散して配置することにより、コンテンツ指向の回路の並列化を行った。

4.3 フレーム指向並列化

また、3.2 節の指紋の持つ時間的連続性に着目した手法でハッシュテーブルに対して導入した、指紋のもつ時間軸の情報に対応するデータ構造を利用して、作成するハッシュテーブルを時間軸方向に分割し、それぞれを 1 つの検索部に対応づけることによって、並列化を行うフレーム指向の並列化を提案する。

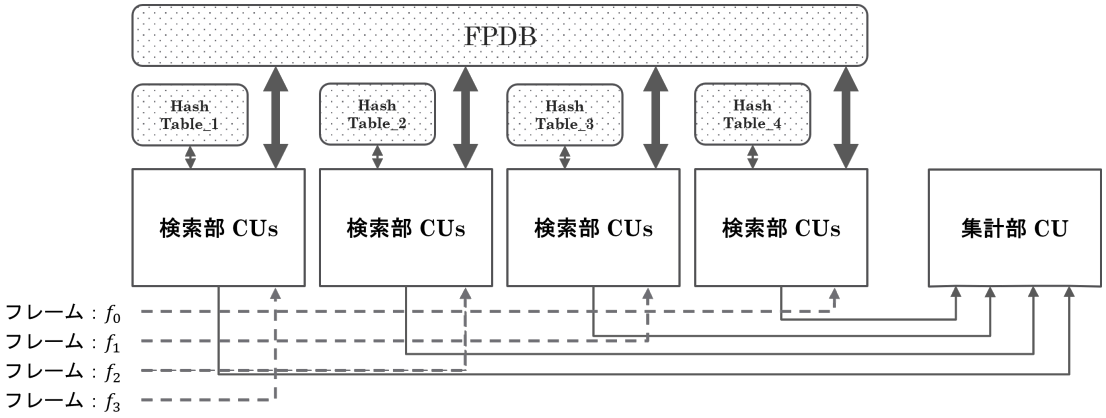

図 4.2: フレーム指向並列化における回路とデータ構造

フレーム指向並列化におけるデータ構造は図 4.2 のようになる。これは、既存のハッシュ関数指向並列化と似たデータ構造を取るため、2.5.4 項の分析からも分かる通り、指紋データベースへのメモリアクセス速度がボトルネックとなることは明らかである。

ここで、既存のハッシュ関数指向の並列化と異なる点は、クエリ指紋から取り出した複数のフレームを同時に処理可能な点にある。表 4.2 に指紋に対して意図的に歪みを与え、指紋連続構造を用いた 1 並列の実装に対して入力した場合の、各 BER における楽曲の特定に要したフレーム数の平均個数を示す。楽曲指紋数 50 万個、試行回数 500 回で各パラメータを表 4.1 に示すように設定して評価を行った。

表 4.1: 評価に用いたパラメータ

パラメータ	設定値
ハッシュ関数 ビット取得数 k bit	8
ハッシュテーブル数 l bit	1
スクリーニング閾値	24
精査閾値	1024

表 4.2: BER による発見までに用いた平均フレーム数の変化

BER	発見に要した平均フレーム数 [個]	BER	発見に要した平均フレーム数 [個]
0.00	1.00	0.14	3.22
0.01	1.09	0.15	3.87
0.02	1.15	0.16	3.93
0.03	1.22	0.17	4.74
0.04	1.39	0.18	4.87
0.05	1.47	0.19	5.57
0.06	1.58	0.20	6.33
0.07	1.75	0.21	6.83
0.08	1.87	0.22	8.27
0.09	2.00	0.23	9.18
0.10	2.38	0.24	11.41
0.11	2.40	0.25	13.55
0.12	2.74	0.26 以上	126.00(未発見)
0.13	3.02		

上記から、歪みの大きな指紋の検索においては、複数のフレームが楽曲の特定までに用いられていることが分かる。一般に 1 並列の実装やコンテンツ指向並列の実装では、楽曲特定までのフレーム数が増えるごとに線形に探索時間が増加する。しかし、フレーム指向の並列化では複数フレームを同時に検索することによって、探索フレーム増加に伴う探索時間増加の傾向を抑制することが可能であると考えられる。各並列化シュッ法において、126 フレームすべてを探索する 1 指紋検索あたりの回路の最悪実行回数 c_f はそれぞれ手法ごとに以下で求められる。ただし、p を並列数とし、p は 126 以下である。

- **1 並列およびコンテンツ指向並列化（p 並列）**

$$c_f = 126 \cdot l \tag{4.1}$$

- **フレーム指向並列化（p 並列）**

$$c_f = \left\lceil \frac{126}{p} \right\rceil \cdot l \tag{4.2}$$

このようにフレームを一度に処理できる利点が、メモリアクセスの競合における速度の低下の欠点を上回ることができれば、歪みの大きな指紋に対して高速な検索が期待できるはずである。このような特性は、歪みの大きな指紋の入力においても実時間で処理される要求のあるシステムにおいて有用であると考えられる。

4.4 関連研究と並列化による速度の期待値

コンテンツ指向およびフレーム指向の並列化によって、検索時間 $t_{parallel}$ は式 (4.3) のようになることが期待される。ここで、t は各検索部ごとの処理時間、p は並列数、 ε は検索部の開始時刻誤差および、集計部の処理時間とする。それぞれの回路の中で最も処理時間の長いものが、1 検索における処理時間となる。

$$t_{\text{parallel}} = \max(t(1), \ldots, t(p)) + \varepsilon \tag{4.3}$$

次に、各カーネルごとの検索時間の期待値について考察していく。文献 [22] では、4 つのデータノードで構成されたクラスタ環境で 100 万曲分の電子指紋を保存したデータベースにおいて、1 並列のものと比較して約 4 倍程度の高速化を達成している。ノードの増加に伴って、データを格納する領域数についても同時に増え、さらに、ソフトウェアによるシーケンシャルな実行であったため、各要素へのアクセスにおいて、競合を考える必要がなく、ほとんど並列数に比例した高速化が可能であった。しかし、今回のハードウェアアクセラレータを用いた実装では、データノード自体を増やすことができるクラスタ環境ではなく、デバイスに実装されているメモリリソースをうまく活用しなければならないため、並列数に比例した高速化は期待できない。

今回、2.5.1 項の図 2.10 で示した構造を持つハードウェアアクセラレータを用いた、1 並列の実装においては、指紋データベースとハッシュテーブルを格納しているメモリを完全に分散させることができたため、要素へのアクセス競合による低速化の影響を考えなくてよかった。しかし、4 並列の実装では、搭載されている並列にアクセス可能なメモリは 4 つであるにも関わらず、格納する要素はそれぞれの検索部に対応した 8 個存在する。これによって、指紋データベースとハッシュテーブルへのアクセスにおいて競合が発生する。スクリーニング部におけるハッシュテーブルへの転送回数は、指紋データベースへの転送回数と同じであり、2.5.5 項から、単一の指紋データベースへ 4 つの回路からのアクセスが集中するとき、転送速度がおよそ半分になったことを考えると、データベースおよび対応するハッシュテーブルの 2 要素へのアクセスの競合では、転送速度はおよそ 3/4 倍になると考えられるので、1 つの検索部において最大でも約 3 倍程度の高速化が期待できる。

また、フレーム指向の並列化においては、$BER = 0.25$ のとき、表 4.2 より 13 個以上のフレームが平均的に検索に用いられているので、4 並列の実装では、およ

そ 4 回程度の回路の実行で発見できることが分かる。ここで、指紋データベースに対して競合が発生する時、4 並列の並列化において指紋データベースからの転送速度がおよそ半分になることから、BER が 0.25 のとき 1 つの検索部において最大約 1.63 倍程度の高速化が期待できる。

4.5 おわりに

本章では、Staged LSH の回路の並列化について 2 種類の手法を考案し、それぞれの手法のシステムにおける狙いについて議論した。また、各手法について関連研究と比較したときの期待値について考察した。

第5章　評価および考察

5.1　はじめに

本章では、第 3 章ならびに第 4 章において提案した、LSH を用いた手法に対する、楽曲電子指紋の時間的連続性に着目した性能の向上のための手法と、並列化手法の評価および、それらの考察を行う。

まず、実際の楽曲から生成した電子指紋に対して、指紋の連続性に着目したデータ構造による楽曲の特定を行い、手法が実世界の課題に対して有効性を持つことを検証する。加えて、圧縮された楽曲から生成された指紋と圧縮前の楽曲から生成された指紋同士の比較により明らかにする、同一楽曲と相違楽曲間の BER の分布から、実世界の著作権管理システムに準拠した各パラメータについての考察を行う。次に、指紋の時間的連続性に着目した手法の速度、メモリ使用量、精度、回路規模について評価及び、考察を行う。さらに、本研究と同じく LSH を用いた指紋の探索手法の性能の底上げを狙った手法である隣接 Staged LSH について、本手法を適用し優位点の評価と手法の拡張性の考察を行う。最後に、FPGA デバイスにおいて、提案した各種並列実装について評価および、考察を行う。

実装環境と評価方法については、2.5.2 項で示したものと同様である。図 5.1 に Staged LSH のデータフロー中の評価範囲を示す。FPGA 上に実装したカーネルの実行を行うコマンドを、処理キューに入力する clEnqueueTask の実行がデバイスで開始したときのデバイスカウンタの値と、終了時の値の差から、時間を計測している。また、並列実装の評価では、各カーネルの開始時刻誤差を含んでいる。速度とメモリ使用量の評価に使用する指紋は、大規模なデータベースを想定するため、乱数生成器 std::random_device を用いて生成したものを用いた。

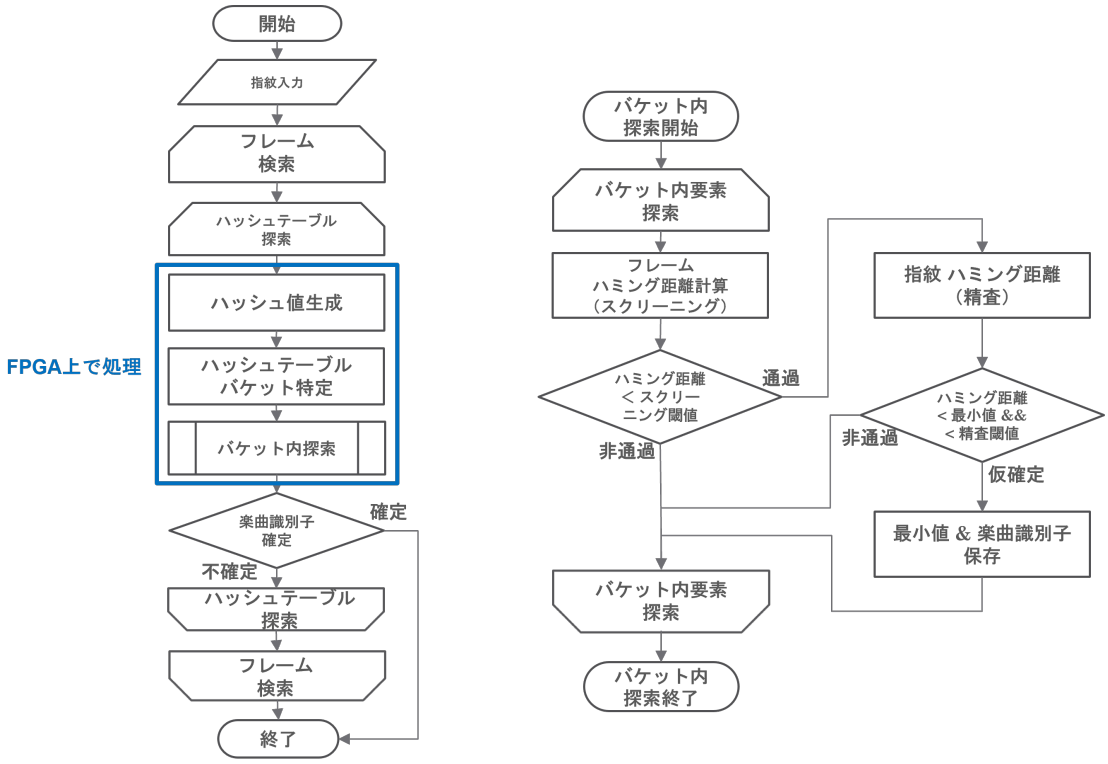

図 5.1: Staged LSH のフローチャートと評価の範囲

5.2 実楽曲による時間的連続性に着目した手法の検証

提案手法が、実際の楽曲から生成した電子指紋を用いた検索において実用性を持っていることを確認するための実験を行った。

表 5.1: mp3 エンコーダ/デコーダ設定

Mp3 encoder / decoder	Lame
Version	Lame 64bit version 3.100
Encoder Option	-b CBR (128 or 192 or 256 or 320)
Decoder Option	- - decode

wav フォーマットで保存された 300 曲のサンプリング周波数 44.1kHz、量子化ビット数 16 ビットのステレオ音源を圧縮前の元楽曲の音源として使用し、これらを頻繁に利用される 128kbps、192kbps、256kbps、320kbps のビットレートで 4 パターンに mp3 の形式で圧縮を行った 1200 曲を用意した。mp3 の圧縮においては、表 5.1 で示したエンコーダを用いた。

そして、用意した非可逆的に圧縮した楽曲と元の楽曲を用いて、2.3 節の Haar ウェーブレット変換のサブバンド分解を用いた手法で指紋を生成した。圧縮前と圧縮後の同一楽曲同士から生成した指紋と、圧縮前と圧縮後の相違楽曲同士から生成した指紋同士の BER を比較して、それぞれの分布を示したのが図 5.2 である。

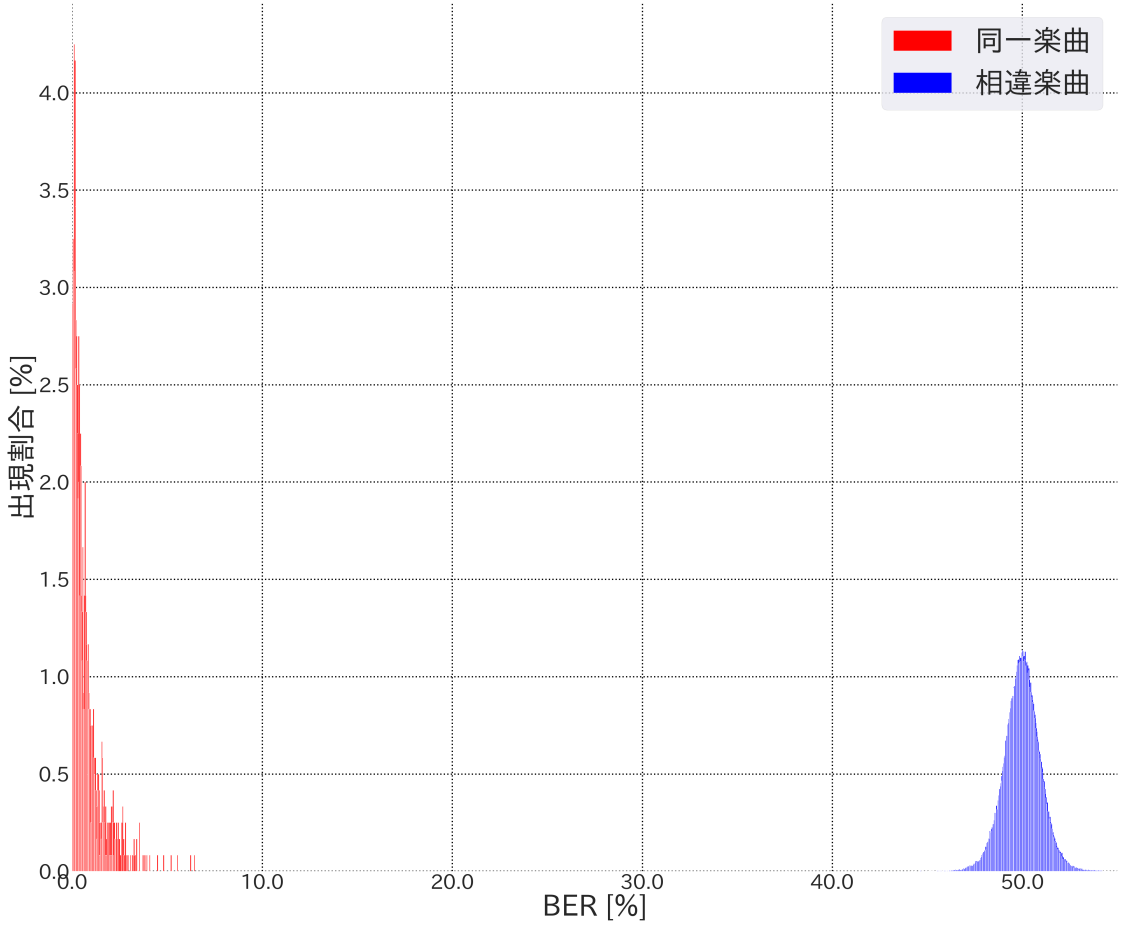

図 5.2: 楽曲から生成した指紋の BER ヒストグラム

同一楽曲から生成した指紋同士の比較では、4 パターンに圧縮した楽曲から生成した指紋と圧縮前の楽曲から生成した指紋同士の比較を行い、1200 パターンの比較を行っている。また、相違楽曲から生成した指紋の比較では、1 曲の圧縮前の楽曲指紋につき、圧縮されたその他の楽曲から生成された指紋と比較したため、89700 パターンの比較を行っている。同一楽曲と相違楽曲間の BER の分布を表すそれぞれのヒストグラムにおける、平均値、分散、標準偏差は表 5.2 のようになった。

表 5.2: 300 楽曲から生成した指紋間の BER 分布分析

	同一楽曲	相違楽曲
平均 BER	0.0070	0.4998
分散	0.65	1.02
標準偏差	0.80	1.01

上記で 300 楽曲を mp3 で 4 パターンに圧縮した楽曲から生成した指紋 1200 個を用いて、スクリーニング閾値を 24、精査閾値を 1024 の各段階における処理要素サイズの 25%に設定した場合、提案手法を用いた実験では楽曲特定の正解率 100%となり、提案手法によって電子指紋により楽曲を特定可能であることを検証した。

5.2.1 誤検出率とパラメータ及び閾値についての考察

Staged LSH を用いた手法にはスクリーニング及び、精査において、閾値を設定している。ここで、5.2 節で求めた実楽曲から生成した指紋間のビットエラーの分布から、精査において用いる閾値の妥当性と 3.3 節で決定したパラメータから求められる誤検出の確率について考察する。ここで、誤検出とは、表 5.3 の検索の結果における誤識別と見逃しの確率を合わせたものである。

表 5.3: 検索結果の区分

		検索結果	
		発見	未発見
実際の結果	発見	OK	見逃し
	未発見	誤識別	OK

各種圧縮により劣化した楽曲がシステムに入力し、あるランダムな数のビットエラーを含む楽曲から生成された指紋が楽曲検索システムに対して入力するとき、あるビットエラーを持つ指紋の入力確率は過去の入力に影響されないため、任意のビットエラー数を持つ指紋の出現確率がポアソン分布に従うと仮定する [24, p.20]。すると、上記で求めた値を用いることで、誤検出率の理論値を求められる。ここで、同一楽曲から生成した指紋同士の BER 分布の平均値を μ_s とする。閾値を精査における要素サイズの割合、BER に設定したとき、指紋データベース内に存在する楽曲と同一楽曲から生成された指紋であるにも関わらず、確率的に見逃して未発見となる確率 P_a は、式 (3.2) を用いて式式 (5.1) で表す。

$$P_a(BER) = \sum_{x=0}^{4096 \cdot BER} \frac{e^{-(4096 \cdot \mu_s)} \cdot (4096 \cdot \mu_s)^x}{x!} \cdot P_f\left(\frac{x}{4096}\right) \tag{5.1}$$

また、指紋データベース内の楽曲と同一楽曲であるにも関わらず、閾値より多くのビットエラーを持つ指紋が生成される確率 P_s は、閾値よりビットエラー数の大きい時のポアソン分布の累積確率の近似値として式 (5.2) で表す。この時、閾値より大きなビットエラー数を持つため、精査の過程で弾かれ最終的な結果は未発見となる。

$$P_s(BER) = \sum_{x=(4096 \cdot BER)+1}^{\infty} \frac{e^{-(4096 \cdot \mu_s)} \cdot (4096 \cdot \mu_s)^x}{x!} \tag{5.2}$$

つぎに、閾値に大きな値を取ることによる、誤識別の可能性を考える。生成する指紋が乱数に近ければ、数百万個程度の指紋の中に相違楽曲にも関わらず、偶然ハミング距離が小さいものが存在する確率は無視できる程度に小さい [25, p.39]。また、指紋データベース内の楽曲に対応するもののない楽曲から生成した指紋が、同一楽曲から生成した指紋以上に類似度が高い時、楽曲を識別するための電子指紋技術としてそもそも破綻しており、同じバケット内の要素であれば最終的な精査処理のハミング距離最小の要素を取り出す過程で、必ず正しい解を発見することができる。そのため、閾値に大きな値を取る時、指紋データベース内に対応する指紋の存在する楽曲指紋が入力されるときには、誤識別の可能性ついてここでは無視できるものとする。ただし、データベース内に対応する楽曲指紋の無いクエリが入力された時、誤識別が発生する確率 P_l は、相違楽曲から生成した指紋同士の BER 分布の平均値を μ_d としたとき、式 (5.3) で表す。

$$P_l(BER) = \sum_{x=0}^{4096 \cdot BER} \frac{e^{-(4096 \cdot \mu_d)} \cdot (4096 \cdot \mu_d)^x}{x!} \cdot \left(1 - P_f\left(\frac{x}{4096}\right)\right) \tag{5.3}$$

以上を用いて、測定したビットエラーの分布から、同一楽曲にも関わらず指紋が閾値以上のビットエラーを持つ場合の未発見率 P_s と、相違楽曲にも関わらず指紋が閾値以下のビットエラーとなる誤識別率 P_l が双方十分に小さく、選択した LSH に関するパラメータによって決まる発見確率 P_a が、最も強く正解率に反映されるときの値をベンチマーク時の閾値として用いるのが妥当だと言える。

表 5.4: 未発見および誤識別の理論値 ($k = 8, l = 1$, $\mu_S = 0.0070$, $\mu_d = 0.4998$)

BER	P_s	P_a	P_l
0.00	< 1.00	0.00	8.33×10^{-890}
0.01	1.71×10^{-2}	3.88×10^{-144}	2.91×10^{-805}
0.02	1.37×10^{-16}	3.97×10^{-120}	2.39×10^{-742}
0.03	1.92×10^{-39}	4.10×10^{-118}	8.20×10^{-689}
0.05	1.23×10^{-106}	4.10×10^{-118}	2.09×10^{-598}
0.10	3.61×10^{-311}	4.10×10^{-118}	1.02×10^{-427}
0.20	2.12×10^{-853}	4.10×10^{-118}	6.78×10^{-210}
0.25	5.13×10^{-1162}	4.10×10^{-118}	1.29×10^{-138}
0.30	1.63×10^{-1488}	4.10×10^{-118}	1.78×10^{-85}
0.40	5.41×10^{-2180}	4.10×10^{-118}	3.62×10^{-21}

表 5.4 の式 (5.2) から求めた値 P_s について、ここではビットエラー数 4096 までの総和を用いた。上記それぞれの近似値から、BER25% において、3.3 節で決定したパラメータによる未発見率 P_a が最も大きな値を取り、なおかつ、同一楽曲間、相違楽曲間の BER 分布のおよそ中心であるため、今回はスクリーニングで用いる閾値を 24、精査で用いる閾値を 1024 とした。なお、この閾値を用いた時、今回の評価において誤識別は含まれなかったため、精度と速度の評価が誤った結果を含まない、手法による効果を示すことを保証する。

最適な閾値やパラメータについては、適応するシステム固有の性質によって、重要視される項目や、要求精度が変わることや、指紋生成手法による頑健性の違いによっても変わるものであるため、一概には決められない。また、それらを踏まえたうえで、確保できるメモリサイズや、要求される速度、検索精度、デバイスに実装されているメモリ数などの様々な制約に対して、多目的な整数線形計画問題を解く必要がある。今回は一定以上の精度で同等であるとして揃えたパラメータを用いることで、手法自体の評価が可能であるため、パラメータ等の最適化手法について、ここでは深く議論せず今後の課題とする。

5.3 時間的連続性に着目した手法の評価

ここでは、3.2 節で提案した時間的連続性に着目した検索手法の評価を行う。特に明記のない限り、入力指紋 $BER = 0.00$ の場合の評価を行っている。

表 5.5: 評価に用いたパラメータ

パラメータ	Staged LSH	Staged LSH+ 指紋連続構造
ハッシュ関数 ビット取得数 k bit	13	8
ハッシュテーブル数 l bit	4	1
スクリーニング閾値	24	
精査閾値	1024	

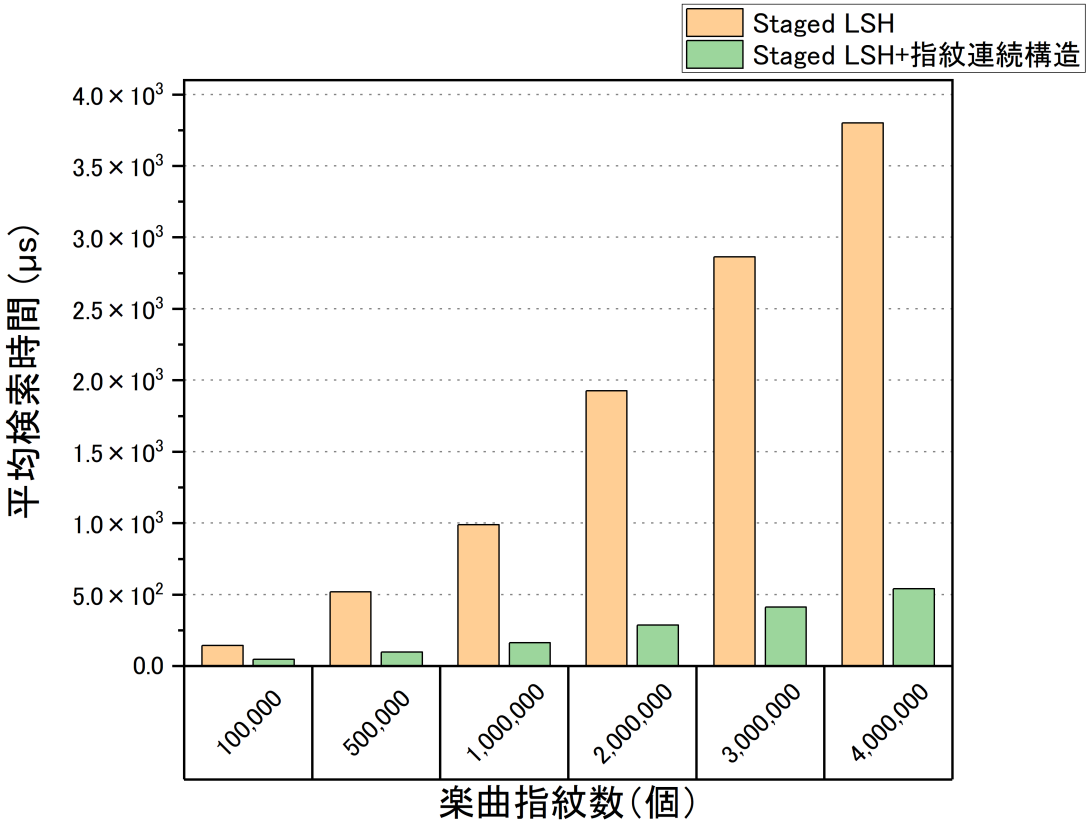

図 5.3: StagedLSH と指紋の持つ連続性に着目した手法適用後の速度比較

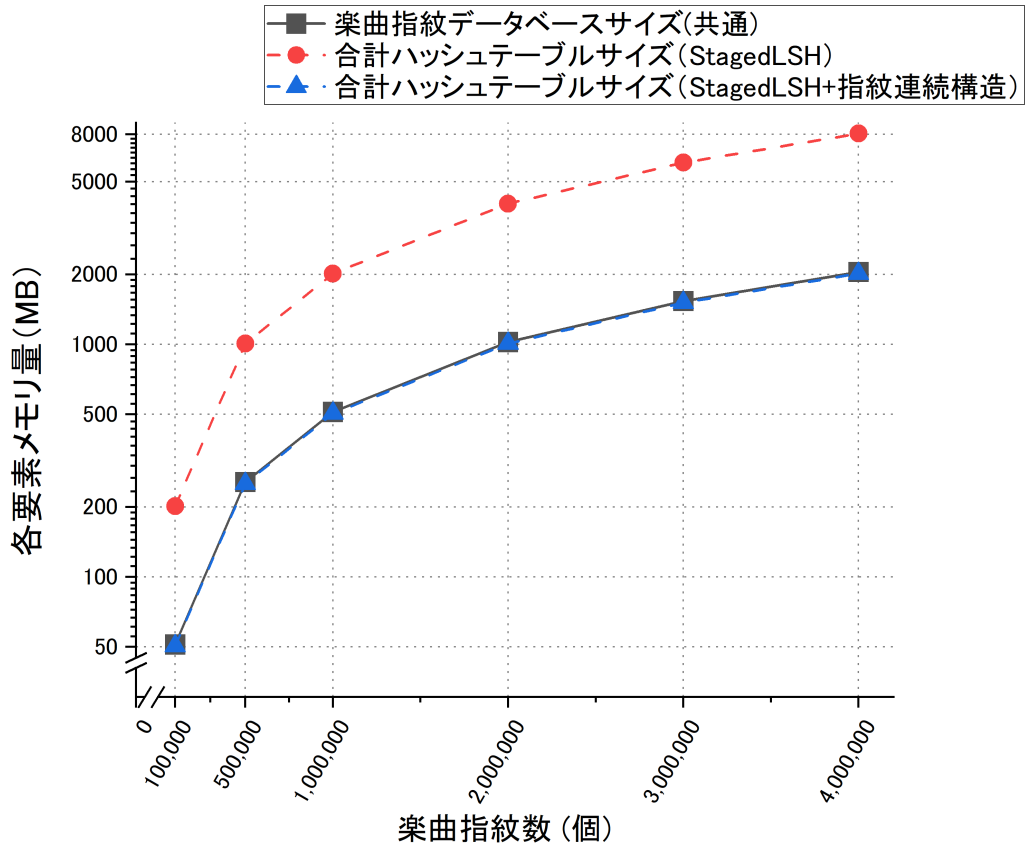

図 5.4: StagedLSH と指紋の持つ連続性に着目した手法適用後のメモリ使用量比較

図 5.3 で示すように 400 万個分の楽曲指紋を対象とした検索では約 7.03 倍の高速化を達成した。理論値の 4 倍程度以上に高速化を達成した理由としては、文献 [12] 中でも言及されていた、ハッシュテーブルにおける要素の分布の不均一さによるものだと考えられる。提案手法では、強制的にフレームの持つ時間情報によってハッシュテーブル内の要素をフレームごとのグループに分割することが可能であるため、理論値に比べ予測以上の高速化が可能であったと考えられる。実際に、スクリーニングカーネル内のデータベースからのフレームの呼び出し回数においても 6~8 倍程度の差がみとめられた。

また、図 5.4 に示すようにハッシュテーブルと楽曲指紋データベースのサイズ比較では、ハッシュテーブルに対するメモリ使用量を約 74.99%ほど抑えることができた。

さらに、楽曲指紋数 10 万個の時、試行回数 10 万回で精度の検証を行ったものが表 5.6 である。パラメータ決定時の想定通り、$BER = 0.25$ のとき 99.99%の精度が得られている。

表 5.6: 実装における各 BER での精度確認（$k = 8, l = 1$）

BER	Staged LSH 正解率 [%]	Staged LSH ＋指紋連続構造 正解率 [%]
0.20	100.000	100.000
0.21	100.000	100.000
0.22	100.000	100.000
0.23	100.000	100.000
0.24	100.000	99.999
0.25	99.999	99.994

また、指紋の持つ時間的な連続性に着目した手法を適用した時、精度が既存の手法を上回るようにパラメータ $k = 8$、$l = 2$ を取る。すると、精度はそれぞれ、表 5.7 のようになり、提案手法のハッシュテーブルの数が 2 倍になったため、総ハッシュテーブルの削減率は、およそ 50%にとどまるが、速度に影響するパラメータ k については、変わっていないため、速度とメモリ効率、精度すべての面で提案手法の有意性が見られる。これらから、提案手法がトレードオフにおける性能底上げに寄与していることを検証できた。

表 5.7: 実装における各 BER での精度確認（$k = 8, l = 2$）

BER	Staged LSH 正解率 [%]	Staged LSH + 指紋連続構造 正解率 [%]
0.20	100.000	100.000
0.21	100.000	100.000
0.22	100.000	100.000
0.23	100.000	100.000
0.24	100.000	100.000
0.25	99.999	100.000

合成時の回路規模はそれぞれ表 5.8 のようになった。LUT はおよそ 9.01%、REG はおよそ 7.50%減少したことが分かる。ハッシュ関数で乗算と加算の処理が増加した分の回路規模の増加に比べて、複数のハッシュテーブルへのインターフェースポート等の削減による回路リソースの削減効果が大きいと考えられる。

表 5.8: StagedLSH 手法と提案手法適応後の回路規模比較

	Staged LSH		Staged LSH ＋指紋連続構造	
Resource	Used Resources	User Budget	Used Resources	User Budget
LUT	10,907	916,750	9,924	916,752
LUT As Mem	1,579	556,935	1,579	556,935
REG	20,564	1,982,310	19,021	1,982,316
BRAM	18	1,603	18	1,603
URAM	0	960	0	960
DSP	0	6,827	0	6,827

また、入力する指紋の持つ BER の変化による検索速度の推移を図 5.5 に示す。5.2 節で求めた、圧縮後および、圧縮前の同一楽曲同士を比較した場合の BER の平均値 0.7%付近では、4.3 節の表 4.2 の BER による平均発見フレーム数の変化からも分かる通り、ほとんどの場合、1 つ目のフレームから発見可能であるため、歪みない指紋の入力と比べても、時間の遅延はほとんどないことが分かった。

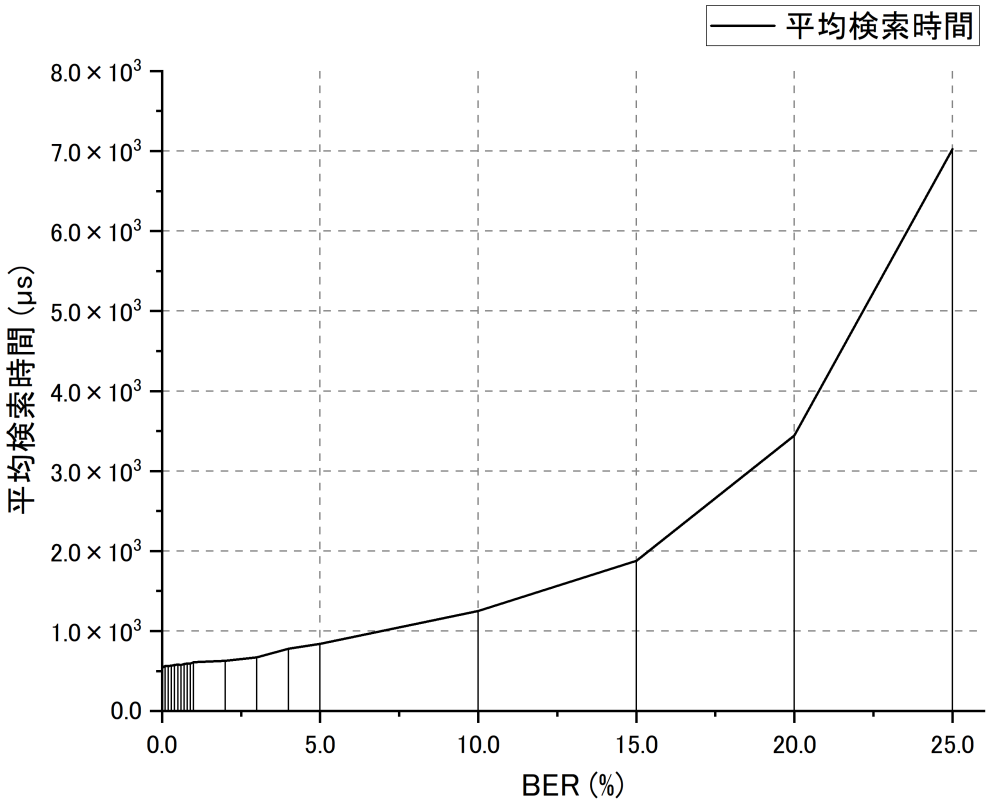

図 5.5: BER の増加による発見速度変化 (指紋数:400 万, 指向回数:500)

5.4 関連研究との比較

次に、隣接 Staged LSH へ、時間的連続性を持たせたデータ構造の適応および評価を行い、Staged LSH を基本とした応用手法への指紋の持つ時間的連続性に着目した手法の拡張性を検証する。ここでは表 5.9 に示す共通のパラメータを取ることで精度を同等とし、評価を行った。

表 5.9: 評価に用いたパラメータ

パラメータ	共通の設定値
ハッシュ関数 ビット取得数 k bit	8
ハッシュテーブル数 l bit	1
スクリーニング閾値	24
精査閾値	1024

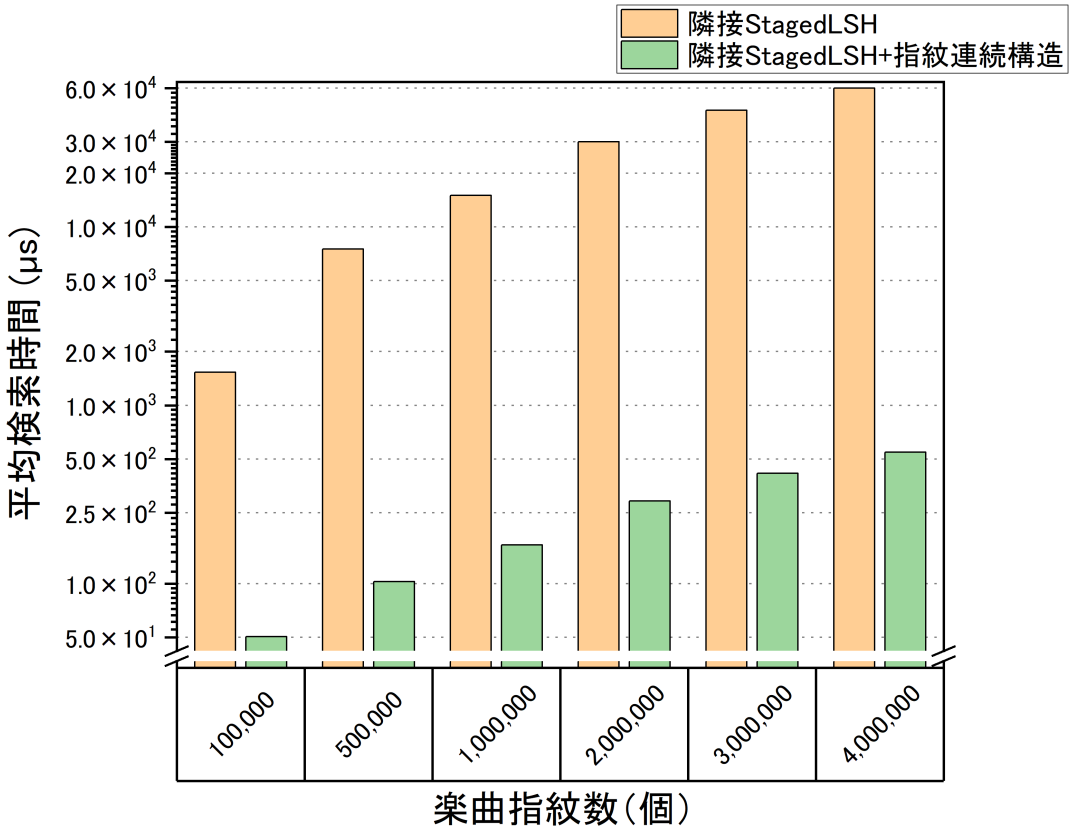

図 5.6: 隣接 StagedLSH と提案手法適応後の速度比較

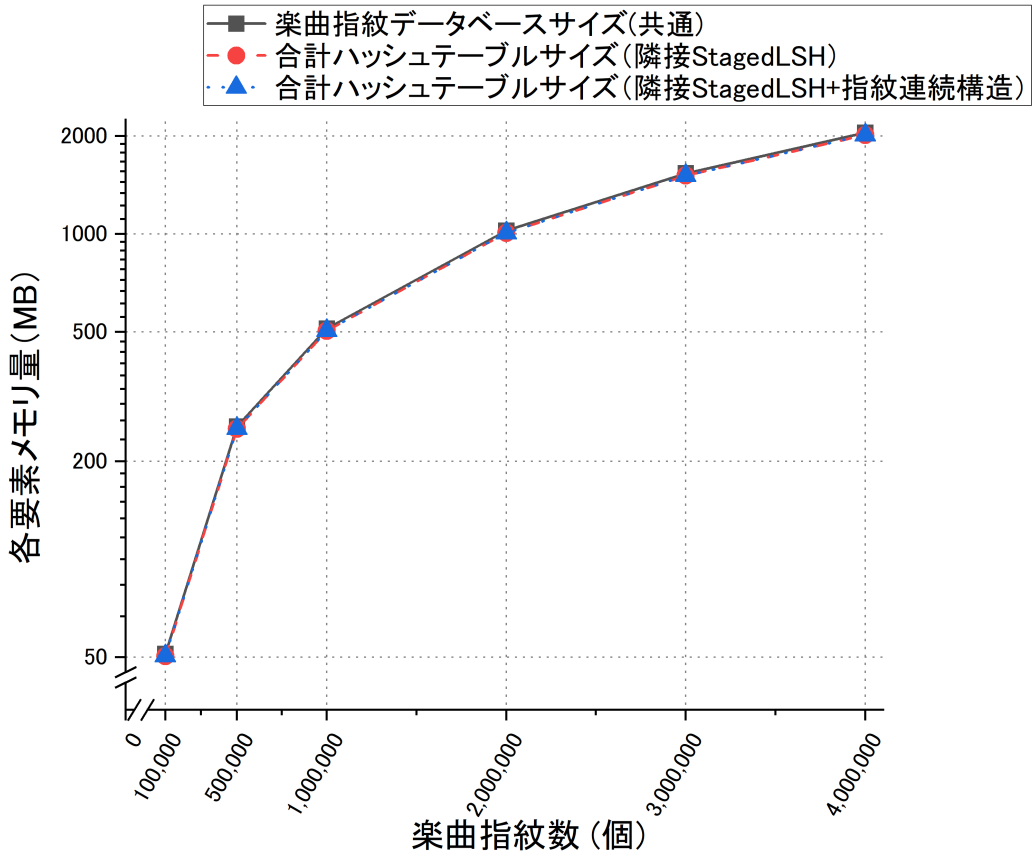

図 5.7: 隣接 StagedLSH と提案手法適応後のメモリ使用量比較

図 5.6 から、100 万個分の楽曲指紋を対象とした検索では約 91.22 倍、400 万個分の楽曲指紋を対象とした検索では約 109.79 倍と楽曲データベース規模が大きくなるほど検索速度の向上が確認できた。今回は、デバイス上のメモリに作成できるバッファサイズに上限があったため 400 万個分の指紋データベースまでを評価の対象としているが、十分に要素を満たしたハッシュテーブルを構成する大規模な楽曲指紋データベースを対象とした検索においても、さらに高速化が可能であると考えられる。

また、両者においてハッシュテーブルの数を決めるパラメータ l を 1 としているため、図 5.7 から分かるように既存の手法においても、提案手法を適応したものにおいてもメモリ効率について、大幅な差はない。ただし、提案手法が値域の拡大分だけハッシュテーブルのバケットを指すインデックスが増加し、わずかだがハッシュテーブルのサイズが増加している。しかし、大規模な楽曲指紋データベースを用いる場合において致命的な差にはならないことが分かった。

5.5 並列化手法の評価および考察

上記で用いた時間軸に着目した検索手法において、第 4 章で考案した並列化のための手法を適応し、評価および考察を行う。ここで、すべての並列化手法において回路を 4 並列とした場合における評価を行っている。

表 5.10: 評価に用いたパラメータ

パラメータ	設定値
ハッシュ関数 ビット取得数 k bit	8
ハッシュテーブル数 l bit	1
スクリーニング閾値	24
精査閾値	1024

5.5.1 各手法の比較

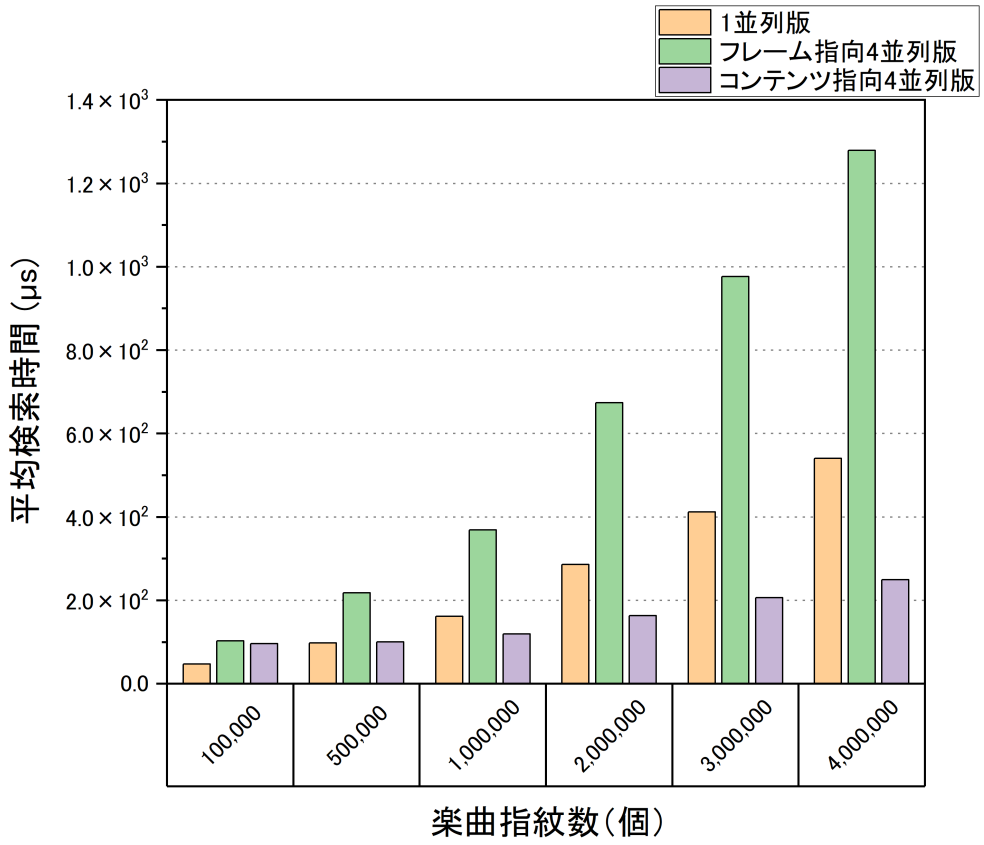

図 5.8: 各並列化手法速度比較（$BER = 0.00$）

表 5.11: 各並列化手法のメモリアクセス解析 (指紋数:100 万, 試行回数:100, $BER = 0.50$)

スクリーニング CU	並列数	平均フレーム転送速度 [MB/s]	平均フレーム転送回数 [回]	最悪 CU 呼び出し回数 [回]
1 並列版	1	371.460	49,235,778	12,600
コンテンツ指向並列版	4	330.386	12,316,843	12,600
フレーム指向並列版	4	176.846	12,494,642	3,200

まず、$BER = 0.00$ の場合の各並列化手法の平均検索時間は図 5.8 のようになった。スクリーニング部の指紋データベースへのアクセスを解析した表 5.11 を見ると、指紋データベースを 4 つの並列アクセス可能なグローバルメモリに対して、分散して格納可能であるコンテンツ指向の並列化では、期待通りにハッシュ関数並

列に比べフレームの転送速度を大きく落とすことなく、転送フレーム数をおよそ 1/4 に削減している。これにより、400 万個の指紋を対象としたデータベースにおいて、1 並列の実装と比較しておよそ 2.16 倍速度の向上が見られた。これは、4.4 節において想定した期待値とも大幅に外れた値とはなっていない。ここで、小さな指紋データベースを対象とした場合の検索において、並列化を施したものが 1 並列のものより速度の面で劣っているのは、各回路の実行開始時刻の誤差や集計部の処理時間、並列に実装した回路の内、最も処理時間の大きい回路の時間が全体の実行時間となる性質などによる遅延が、平均検索時間に占める割合として大きいためであると考えられる。

フレーム指向の並列化においては、1 回路におけるフレームの最悪転送回数は 1 並列のものにくらべおよそ 1/4 となっているが、回路の最悪実行回数自体も 1/4 回であるため、1 度の回路の実行において指紋データベースからの要素の転送回数は 1 並列のものと等倍である。そのため、想定通りに下がったフレームの転送速度の影響により、1 並列の回路と比較しても速度の低下がみられる。

しかし、フレーム指向並列化の意図としては、複数フレーム同時検索による歪みの大きな指紋の検索においての高速化である。そこで、今回実装している回路において発見することができる最大の BER である 0.25 のビットエラーを持つ指紋に対して速度の評価を行った。劣化の小さい楽曲から生成されると想定される指紋の検索においては、400 万個の楽曲指紋を対象としても数ミリ秒以内での検索が可能であることが分かったため、劣化の大きな楽曲から生成されることが予測される指紋に対する検索時間の評価を行うことで、フレーム指向並列化の優位点を検証する。図 5.5 から、入力される指紋の BER が大きくなるほど、検索速度が遅くなることが分かるため、想定する BER として大きすぎるとも思われる $BER = 0.25$ の場合の指紋検索について高速化できれば、システムに入力されるクエリ全てにおいて、一定時間内に処理可能な指紋データベース規模の拡張が期待できる。

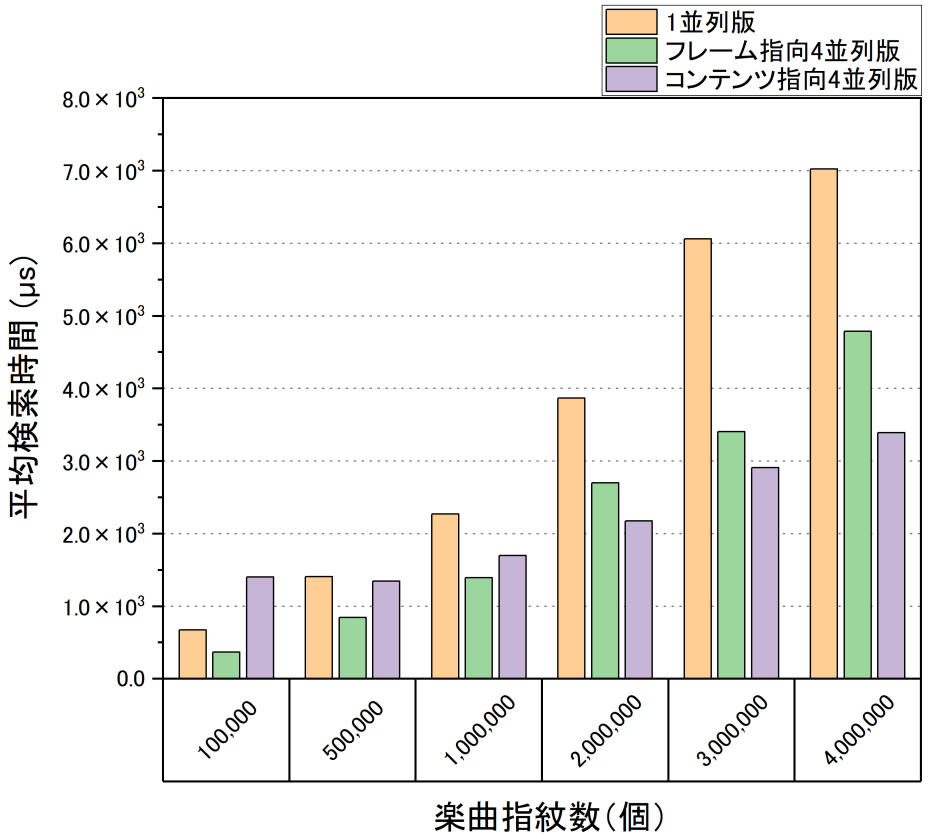

図 5.9: 各並列化手法速度比較（$BER = 0.25$）

図 5.9 より、検索対象の指紋の持つビットエラー数の大きい（$BER = 0.25$）とき、フレーム指向並列化によって、1 並列の実装より約 1.47 倍検索時間を短縮できることが分かった。これは、4.4 節により考察した高速化の期待値とおよそ合っている。この結果から、複数のフレームを一度に検索できる利点が、指紋データベースへのアクセス速度低下の利点に上回ったことが分かる。

しかし、コンテンツ指向の並列化を行った場合はこれよりさらに高速な検索が可能であることも同時に分かった。300 万個と 400 万個の指紋データベースを対象とした場合の、フレーム指向並列化とコンテンツ指向並列化の検索時間を比較すると、その差が大きくなっていることから、今後、指紋データベース規模が大きくなるに従って、さらにコンテンツ指向の優位性が際立つと考えられる。

5.5.2 フレーム指向並列化の有効点

上記で、検索対象の指紋が大きなビットエラーを含む場合においてもコンテンツ指向による並列化が有効であることが分かった。しかし、コンテンツの分散を主軸とした並列化には、デバイス上にハッシュテーブルと指紋データベースを分散して配布するための並列にアクセス可能な複数メモリを持つことが条件に挙げ

られる。つまり、今回実装に用いたような複数のメモリ資源のある環境では、利用可能なメモリの数に伴って手法が高速化に威力を発揮するが、様々な制約上、同時にアクセス可能な複数のメモリが存在しない場合には性能の低下が懸念される。

ここで、今回用いたプラットフォーム構成以外の、並列にアクセス可能なメモリの限られる環境における手法の汎用性を確認しておきたい。そこで、要素を格納するグローバルメモリ数を制限した環境において、フレーム指向による並列化の利点がどれだけ優位に働くか検証した。

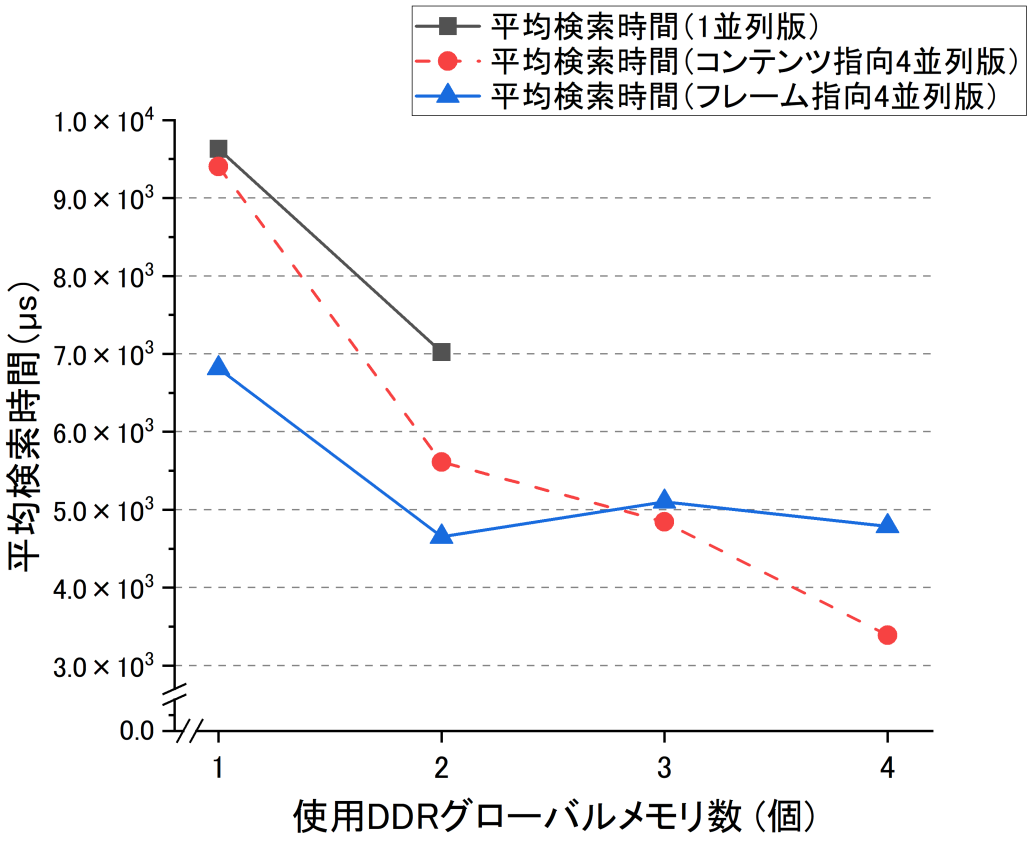

図 5.10: 使用メモリ数による検索速度の遷移（指紋数:400 万,$BER = 0.25$）

表 5.12: 各並列化手法と要素の格納に用いたメモリ数ごとのメモリアクセス解析 (指紋数:100 万, 試行回数:100, $BER = 0.50$)

スクリーニング CU	使用メモリ数	平均フレーム転送速度 [MB/s]	平均フレーム転送回数 [回]	最悪 CU 呼び出し回数 [回]
1 並列版	1	277.808	49,223,822	12,600
	2	371.460	49,235,778	12,600
コンテンツ指向 4 並列版	1	143.983	12,319,160	12,600
	2	303.924	12,320,079	12,600
	3	284.642	12,317,741	12,600
	4	330.386	12,316,843	12,600
フレーム指向 4 並列版	1	120.450	12,501,113	3,200
	2	192.895	12,498,759	3,200
	3	179.431	12,502,354	3,200
	4	176.846	12,494,642	3,200

図 5.10 の各要素を格納するのに用いたグローバルメモリの数の変化と、それに伴う歪みの大きな指紋検索時間の遷移から、フレーム指向並列化は並列にアクセス可能なメモリの数が限られる場合、歪みの大きな指紋の検索において有意点が存在することが分かる。スクリーニング部の指紋データベースへのアクセスを解析した表 5.12 からも、使用メモリ数 1 におけるコンテンツ指向並列化のフレームの転送速度低下がみとめられる。1 つのグローバルメモリに対して、複数のフレーム呼び出しを行うため、複数フレームを 1 度に検索可能なフレーム指向による並列化に優位性が生まれたものだと分かる。

つまり、フレーム指向並列化は、個々のメモリの持つ容量ではなく、デバイス上に実装されているメモリそのものがコストまたは、サイズ等の何らかの制約によって制限される状況において、歪みの大きな指紋の検索を含め、一定時間内で検索可能な指紋データベース規模の拡大を図ることができる手法であると言える。

5.6 おわりに

本節では、第 3 章および、第 4 章で提案した各種手法について評価を行った。まず、実楽曲から生成した指紋を用いた評価により、本手法が実世界における電子指紋を用いた楽曲の検索に有用であることを検証した。また、楽曲電子指紋の時間的連続性に着目した手法の評価から、400 万個の指紋を格納したデータベースにおいて、3.3 節で選んだ同等精度のパラメータでは、約 74.99%のハッシュテーブルの削減と、約 7.03 倍の検索の高速化を達成した。加えて、LSH の拡張手法である隣接 Staged LSH においても、精度、メモリ効率を同等としたとき、約 109.79 倍

の高速化を達成した。以上より、手法がLSHを用いた楽曲電子指紋の検索において性能の底上げが可能であることを示した。最後に、並列化手法について、4並列で実装したコンテンツ指向並列化と1並列で実装したものとを比較して、約2.16倍の高速化を達成した。結果から、コンテンツ指向による並列化が基本的な並列化戦略としては有効であることが分かった。また、同時にアクセス可能なメモリ数の制約がある環境におけるフレーム指向並列化の優位点についても検証した。

第6章　まとめと今後の課題

6.1　結論

はじめに、楽曲から生成される電子指紋を用いた楽曲の検索において、類似した指紋を発見する手法である、Staged LSH に対して、指紋の持つ時間的な連続性に着目したハッシュテーブルのデータ構造並びに、格納、検索手法の改良を行った。著作権管理システムなどでのシステムの運用を仮定し、SSD などの外部ストレージに比べ、高速にアクセス可能なグローバルメモリ上に、可能な限り大規模な指紋データベースを格納したい要求や、組込み機器等のメモリサイズに制約の大きい環境を考慮し、メモリ空間効率の向上を優先した、既存の手法と同等検索精度となるパラメータを選択して手法の評価を行った。評価の結果、400 万個の指紋を格納した楽曲指紋データベース上で、時間的な連続性に着目した手法を用いて約 74.99%の総ハッシュテーブルサイズの削減と約 7.03 倍の高速化を達成した。また、Staged LSH の拡張手法である隣接 Staged LSH に対して、精度、ハッシュテーブルサイズ共に同等のパラメータを選んだ時、約 109.79 倍の高速化を達成した。これらから、LSH を用いた指紋の検索手法における精度、速度、総ハッシュテーブル容量におけるトレードオフに対して、提案手法により性能の底上げが可能であることが分かった。

また、処理の高速化を目的とした FPGA デバイス上の回路並列化において、デバイス上のグローバルメモリを主要な要素格納領域として利用する際、既存のハッシュテーブル間の独立性に着目したハッシュ関数指向の並列化における課題を分析し、指紋データベースへのアクセス競合による検索速度低下の問題から、データベースの分割によるコンテンツ指向の並列化を考案した。コンテンツ指向並列化で 4 並列に回路を実装したところ、400 万個の指紋を格納した楽曲指紋データベースにおいて、1 並列の場合と比較した時、約 2.16 倍の高速化を達成した。加えて、今回提案した、指紋の時間的連続性に着目した LSH の拡張手法に対応した、ハッシュテーブルの構造を応用して、フレーム指向のハッシュテーブルの分割による回路の並列化を考案した。評価の結果から、デバイス上に複数の並列にアクセス可能なメモリが実装されている環境においては、それぞれに指紋データベースを分割し、それぞれに対応したハッシュテーブルを用いる、コンテンツ指向の並列化がフレーム指向の並列化に比べても高速化に有効であることが分かった。しかし、同時にアクセス可能なメモリの数自体に制限のある環境において、ビットエラー率 25%のような、歪みの大きな指紋の入力に対する検索を行うとき、フレー

ム指向の並列化がコンテンツ指向並列化と比較しても高速化に優位性を持つことが分かった。

以上から、高い精度で類似指紋を発見する検索手法に対して、速度とメモリ効率の面で改善を行ったことで、一定時間内に処理可能な指紋データベース規模を拡張したことに加え、限られたメモリ空間に格納可能な指紋データベース規模の拡張を行った。また、今回の提案手法に対して、楽曲の流行などを考慮し、回路内の高速な RAM をうまく利用する手法を併用することで、パレートの法則により偏りがあると考えられる、実世界のクエリに対して、さらに処理の高速化が期待できると考えられる。

6.2 今後の課題

今後の課題、展望として主に以下の 2 点について改良や考察の余地があると考える。

6.2.1 メモリ帯域幅の有効活用による高速化

今回用いた Alveo U200 アクセラレーションカードでは、グローバルメモリとカーネル間でやり取りされる最大データ幅は 512 ビットであるにも関わらず、今回対象としている手法では検索単位として一部前後の要素と重複のある 96 ビットのフレームを用いているため、帯域幅を有効活用することができていない。コード 6.2.1 によると、FPGA に実装された回路とメモリ間のスループットは 48075MB/s 程度だが、今回の手法であればスクリーニング部の指紋の呼び出しにおいて高くとも 400MB/s 程度しか利用されていない。そこで、データベースから、指紋を効率的に読みだすことのできる手法を提案することで、さらなる処理の高速化が可能だと考えられる。

コード 6.2.1 Alveo U200 の xbutil validate コマンドによる性能確認

```
INFO: Found 1 cards

INFO: Validating card[0]: xilinx_u200_xdma_201830_2
INFO: == Starting Kernel version check:
INFO: == Kernel version check PASSED
INFO: == Starting AUX power connector check:
INFO: == AUX power connector check PASSED
INFO: == Starting Power warning check:
INFO: == Power warning check PASSED
INFO: == Starting PCIE link check:
INFO: == PCIE link check PASSED
INFO: == Starting SC firmware version check:
INFO: == SC firmware version check PASSED
INFO: == Starting verify kernel test:
INFO: == verify kernel test PASSED
INFO: == Starting DMA test:
Host -> PCIe -> FPGA write bandwidth = 8294.419492 MB/s
Host <- PCIe <- FPGA read bandwidth = 13489.836515 MB/s
INFO: == DMA test PASSED
INFO: == Starting device memory bandwidth test:
...........
Maximum throughput: 48075 MB/s
INFO: == device memory bandwidth test PASSED
INFO: == Starting PCIE peer-to-peer test:
P2P BAR is not enabled. Skipping validation
INFO: == PCIE peer-to-peer test SKIPPED
INFO: == Starting memory-to-memory DMA test:
bank0 -> bank1 M2M bandwidth: 11944.8 MB/s
bank0 -> bank2 M2M bandwidth: 11959.3 MB/s
bank0 -> bank3 M2M bandwidth: 11962.1 MB/s
bank1 -> bank2 M2M bandwidth: 11958.7 MB/s
bank1 -> bank3 M2M bandwidth: 11958.7 MB/s
bank2 -> bank3 M2M bandwidth: 11958.7 MB/s
INFO: == memory-to-memory DMA test PASSED
INFO: == Starting host memory bandwidth test:
Host_mem is not available. Skipping validation
INFO: == host memory bandwidth test SKIPPED
INFO: Card[0] validated successfully.

INFO: All cards validated successfully.
```

6.2.2 演算器稼働率向上による高速化

本研究で対象としているシステムの最終目的はデジタル著作物の利用活性化であるため、検索におけるクエリの大半を楽曲指紋データベースに格納されているコンテンツが占めることを前提としている。しかし、悪意を持ったユーザによって意図的に、もしくは、著作物を取り締まるシステムのような、システムの利用されるサービス自体の傾向として、著作権管理システムで管理されていない楽曲が入力される機会がクエリの大半を占める場合、検索における発見時の時間と比較して、特定の楽曲が見つからない時の最悪実行時間がシステムを構築する上で重要視される場合がある。

このような場合、各フレームの検索ごとに同期をとり検索終了であるか判定を行うシステムではなく、ハードウェアの実装に対して、ストリームデータ処理を

用いて非同期的なフレーム間の処理を行うことで、回路の稼働率を最大とし、システムにおける平均的な検索処理の高速化が可能だと考えられる。

研究業績

- 野本 健心, 井口 寧, ”楽曲電子指紋の時間的連続性に着目した省メモリ化とデータベースの実時間検索ハードウェアアクセラレーション”, 2022 年度電気・情報関係学会北陸支部連合大会

謝辞

本論文の作成にあたり、日々多くの助言を賜り、ご指導ご鞭撻のほど頂いた井口 寧教授に深謝の意を表します。また、審査会や合同ゼミにおいて、研究の方針についてご助言、ご指導いただいた方々にも厚く御礼申し上げます。最後に、井口研究室の皆様には本研究の遂行にあたり、常日頃から多くのご助力を頂きましたこと、ここに深く感謝申し上げます。

参考文献

[1] 平成28年度著作権委員会第4部会. “デジタルコンテンツの保護と利用についての研究.” パテント = patent, vol. 70. no. 12 (Nov. 2017), pp. 74–91. ISSN: 02874954. URL: https://cir.nii.ac.jp/crid/1523669555326851840.

[2] 渡辺 保史. “デジタルコンテンツの知的所有権ー Digital Rights (O’REILLY COMMUTER SERIES).” オライリー・ジャパン, Mar. 1998, p. 280. ISBN: 978-4900900530.

[3] 株式会社インプレス. “定額制音楽配信サービスの利用に関する調査結果2018.” Tech. rep. 2018. URL: https://www.impress.co.jp/newsrelease/pdf/20180330-01.pdf.

[4] “JOYSOUND.com.” 株式会社エクシング. URL: joysound.com/web/search.

[5] “Uta-Net.” 株式会社ページワン. URL: https://www.uta-net.com/.

[6] Asif Ghias, Jonathan Logan, David Chamberlin, and Brian C. Smith. “Query by Humming: Musical Information Retrieval in an Audio Database.” *Proceedings of the Third ACM International Conference on Multimedia*. MULTIMEDIA ’95. San Francisco, California, USA: Association for Computing Machinery, 1995, pp. 231–236. ISBN: 0897917510. DOI: 10.1145/217279.215273. URL: https://doi.org/10.1145/217279.215273.

[7] Erdem Unal, Elaine Chew, Panayiotis G. Georgiou, and Shrikanth S. Narayanan. “Challenging Uncertainty in Query by Humming Systems: A Fingerprinting Approach.” IEEE Transactions on Audio, Speech, and Language Processing, vol. 16. no. 2 (2008), pp. 359–371. DOI: 10.1109/TASL.2007.912373.

[8] Mani Malek Esmaeili, Mehrdad Fatourechi, and Rabab Kreidieh Ward. “A Robust and Fast Video Copy Detection System Using Content-Based Fingerprinting.” IEEE Transactions on Information Forensics and Security, vol. 6. no. 1 (2011), pp. 213–226. DOI: 10.1109/TIFS.2010.2097593.

[9] 田中 賢一. “電子透かし技術ーディジタルコンテンツのセキュリティ.” Ed. by 画像電子学会. 東京電機大学出版局, Jan. 2004, p. 220. ISBN: 978-4501323202.

[10] Kelkoul Houria, Youssef Zaz, and Teddy Mantoro. “Hybrid Watermark Fingerprint Algorithm for H.264 Compressed Video Authentication.” *2018 International Conference on Computing, Engineering, and Design (ICCED)*. 2018, pp. 250–253. DOI: 10.1109/ICCED.2018.00056.

[11] 天羽健介 and 増田雅史. “NFT の教科書　ビジネス・ブロックチェーン・法律・会計まで　デジタルデータが資産になる未来.” 朝日新聞出版, Oct. 20, 2021, p. 381.

[12] Jaap Haitsma and Ton Kalker. “A Highly Robust Audio Fingerprinting System.” *ISMIR*. 2002. URL: http://dblp.uni-trier.de/db/conf/ismir/ismir2002.html#HaitsmaK02.

[13] 荒木 光一, 幸紀 佐藤, V.K. Jain, and 井口 寧. “ハードウェアにおける高速なオーディオフィンガープリント 生成システムの性能評価.” 先進的計算基盤システムシンポジウム: SACSIS 2010 論文集, vol. 2010. no. 5 (May 2010), pp. 295–302. ISSN: 1344-0640. URL: https://cir.nii.ac.jp/crid/1050574047132247168.

[14] ジェイムズ ノーブル, チャールズ ウィアー, James Noble, and Charles Weir. “省メモリプログラミングーメモリ制限のあるシステムのためのソフトウェアパターン集 (Software patterns series).” Trans. by 慶一 安藤. ピアソンエデュケーション, June 20, 2002, p. 402. ISBN: 978-4894714083.

[15] Piotr Indyk and Rajeev Motwani. “Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality.” *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. STOC ’98. Dallas, Texas, USA: Association for Computing Machinery, 1998, pp. 604–613. ISBN: 0897919629. DOI: 10.1145/276698.276876. URL: https://doi.org/10.1145/276698.276876.

[16] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. “Similarity Search in High Dimensions via Hashing.” *Proceedings of the 25th International Conference on Very Large Data Bases*. VLDB ’99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 518–529. ISBN: 1558606157.

[17] Fan Yang. “Hardware Acceleration of Searching Strategy for Audio Fingerprinting System.” *Thesis or Dissertation*. June 2013. URL: http://hdl.handle.net/10119/11398.

[18] 井口 寧 and 福田 真啓. “FPGA による電子指紋検索の高速化事例.” *DA* **シンポジウム** *2017* **論文集**. vol. 2017. Aug. 2017, pp. 103–108.

[19] “Alveo データセンターアクセラレータカードプラットフォーム ユーザーガイド.” XILINX, June 2020. URL: https://japan.xilinx.com/content/dam/xilinx/support/documents/boards_and_kits/accelerator-cards/j_ug1120-alveo-platforms.pdf.

[20] 天野英晴, ed. “FPGA の原理と構成.” オーム社, Apr. 22, 2016, p. 292. ISBN: 978-4274218644.

[21] Zeke Wang, Johns Paul, Bingsheng He, and Wei Zhang. “Multikernel Data Partitioning With Channel on OpenCL-Based FPGAs.” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25. no. 6 (2017), pp. 1906–1918. DOI: 10.1109/TVLSI.2017.2653818.

[22] Ce Yu, Runtao Wang, Jian Xiao, and Jizhou Sun. “High performance indexing for massive audio fingerprint data.” IEEE Transactions on Consumer Electronics, vol. 60. no. 4 (2014), pp. 690–695. DOI: 10.1109/TCE.2014.7027344.

[23] “Vitis 高位合成ユーザーガイド.” XILINX, Aug. 2021. URL: https://www.xilinx.com/content/dam/xilinx/support/documents/sw_manuals_j/xilinx2021_1/ug1399-vitis-hls.pdf.

[24] 島谷 健一郎. “ポアソン分布・ポアソン回帰・ポアソン過程 (統計スポットライト・シリーズ).” 近代科学社, Oct. 23, 2017, p. 124. ISBN: 978-4764905467.

[25] 福田 真啓. “大規模な音楽指紋データベースの高速検索におけるクエリの歪みへの頑健性向上に関する調査研究.” *Thesis or Dissertation*. Sept. 2015. URL: http://hdl.handle.net/10119/12925.