Vol.2016-CDS-16 No.6
2016/6/2

車載地図更新向け列指向データ圧縮の高速化

関口隆昭[†1]　永井靖[†1]　名越末男[†2]　正嶋博[†2]　福永良一[†2]

概要：カーナビのストレージに格納している地図データの圧縮方式について述べる．地図データの鮮度に対する要求は年々高まっているが，データ更新を容易に行うためには地図データのサイズを削減する必要がある．本研究では，圧縮データの伸張処理によるナビ性能低下をユーザが感じないよう，地図データの多くを占める固定長レコードを対象に高速伸張可能な圧縮方式を提案する．提案方式は，RDBMS で普及しつつある列指向データ圧縮をベースに，キャッシュ効率化と複数の軽量な符号化方式を利用することで，Zlib 相当の圧縮率と非圧縮時同等のリード性能実現を図ったものである．提案方式をカーナビに実装し，特定のデータに対する上記性能の実現性を確認した．

キーワード：地図更新，地図データ，列指向データ圧縮，カーナビ

Fast Columnar Data Compression for In-Vehicle Map Update

TAKAAKI SEKIGUCHI[†1]　YASUSHI NAGAI[†1]
SUEO NAGOSHI[†2]　HIROSHI SHOJIMA[†2]　RYOICHI FUKUNAGA[†2]

Abstract: In this paper, we describe a data compression method for map data in car navigation systems. There is a growing need for up-to-date map data. But in order to update it easily, the data size should be reduced. We propose a fast compression method designed especially for fixed-length records consisting mostly of map data for users not to feel performance degradation by the decompression process. The method is based on columnar data compression technology mainly used in RDBMS. We tried to realize Zlib-equivalent compression ratio and high throughput similar to that of uncompressed data by considering cache efficiency and several light-weight coding algorithms. We implemented the method on a system and evaluated its performance. The presented result shows feasibility of the target performance in particular data.

Keywords: map update, map data, columnar data compression, car navigation system

1. はじめに

カーナビゲーションシステム用地図データの鮮度や精度に対する要求が年々高まっている．スマートフォンの地図アプリではユーザが特に意識しなくても常に最新地図を利用可能であり，これと同等のことがカーナビにも求められている．またクラウド上の様々な地図サービスとカーナビの連携が進む中，カーナビの地図データが古いと適切な案内を行えないことも多い．さらに近年では自動車の制御にカーナビの地図データを活用することも検討されており，データの鮮度や精度が一層重要になっている．

このような要求に対し，従来のカーナビは内部のストレージに格納している道路や施設の情報（地図データ）を更新しにくいという問題がある．主な理由は地図データのサイズであり，数 GB ある地図データを 3G 等の無線通信でダウンロードして更新することは，更新時間や通信費の点でユーザの負担が大きいからである．そのため現在は，地図データを更新するために SD カード等の記録メディアをカーナビから取り外して自宅の PC で書き換えたり，販売店に更新作業を依頼する等の手段がとられている．

本稿では，地図データの更新を容易化するための取組みの一環として，カーナビ内部のストレージに格納している地図データの圧縮方式について述べる．高速に伸張可能な方式で地図データを圧縮することにより，最新地図データのダウンロード時間やストレージ使用量の削減が可能になり，また，圧縮によって空いた領域に従来より高精度なデータを格納することが可能になる．本稿で提案する方式は，RDBMS 等で普及しつつある列指向データ圧縮をベースに，地図データの多くを占める固定長レコードを圧縮する際の圧縮率と伸張性能の向上を図ったものである．

以下，2 章で関連研究，3 章で解決すべき課題を説明する．4 章において提案方式の詳細を述べ，5 章で提案方式を実装して評価した結果を述べる．最後に 6 章で纏める．

2. 関連研究

2.1 列指向データ圧縮

圧縮対象とするデータが RDB のように行と列の二次元の構造を有する場合，列毎に圧縮すると入力データの系列に特徴が現れるため圧縮率を向上できることが知られている[1]．このように列毎に圧縮する方式は列指向データ圧縮と呼ばれている．

列指向データ圧縮に関する従来の取組みとして C-Store がある[2]．Abadi 等は C-Store を対象に列指向データ圧縮と

†1 株式会社日立製作所　研究開発グループ
Hitachi, Ltd. Research & Development Group
†2 クラリオン株式会社
Clarion Co., Ltd.

ⓒ2016 Information Processing Society of Japan

クエリ性能の最適化に取組み，列の特性に応じて符号化方式を切り替える方式について述べている[3]．C-Store ではランレングス符号，辞書符号，ビットベクタ符号，LZ 符号[4]の 4 種類を評価した結果，ランレングス符号等の軽量な符号化を用いた場合の性能が良いとしている．

また，郡等はデータウェアハウス向け RDBMS への列指向データ圧縮技術の適用に取組み，ランレングス符号，インデックス符号，差分符号を組み合わせた RID 符号を開発している[5]．RID 符号は，伸張性能を確保するために上記の軽量な符号のみの組合せによって実現しており，また，対象データを一度走査し，列の特性に応じてインデックス符号のビット長や差分符号の適用有無を切り替えることにより高い圧縮率を達成している．

2.2 PAX レイアウト

前述のように，列指向でデータを圧縮することにより特定の列のみを参照するクエリの性能を向上できる一方，全ての列を参照する処理の場合，例えばレコード 1 行を伸張するような場合は，列毎に圧縮された全てのデータを伸張してデータを取り出す必要があるため逆に性能が低下する．このような列指向でデータを処理する際のトレードオフに関連する取組みとして，RDBMS 向けのデータ配置モデルである PAX（Partition Attributes Across）が知られている[6][7]. PAX 自体はデータ圧縮に関するものではないが，本稿で提案する方式と関係するため，ここで説明する．

RDBMS のデータをストレージに記録する際のモデルとして，レコード順にデータを記録する NSM（N-ary Storage Model）と，列毎にデータを分割して記録する DSM（Decomposition Storage Model）がある[8][9]．検索クエリが参照する列は数個のみであることが多いため DSM のほうが良い性能を示すことが多いが, DSM は参照する列が多くなると性能が低下する．PAX はこのトレードオフに対する提案であり, DSM のようにレコードを列毎に分割して格納しつつ，データリード用のキャッシュ 1 ページに全ての列が収まるように，データを一定のレコード数で分割して格納する（図 1）．これによって，例えば特定の列のみを参照するクエリを高速化しつつ，多くの列を参照するクエリについても性能低下を抑えている．

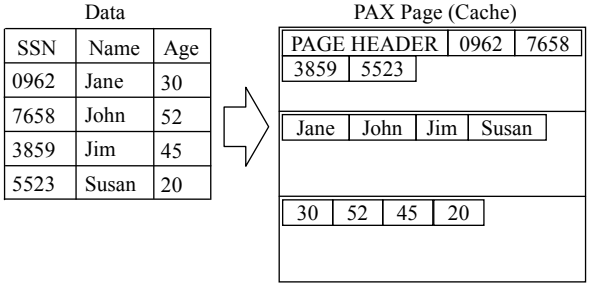

図 1　PAX によるデータ配置
Figure 1　Storage layout by PAX.

3. 課題

3.1 要件

列指向データ圧縮を用いて地図データを高速伸張可能なように圧縮することを考える．図 2 に，地図データ更新システムをデータ圧縮の観点で分類したものを示す.

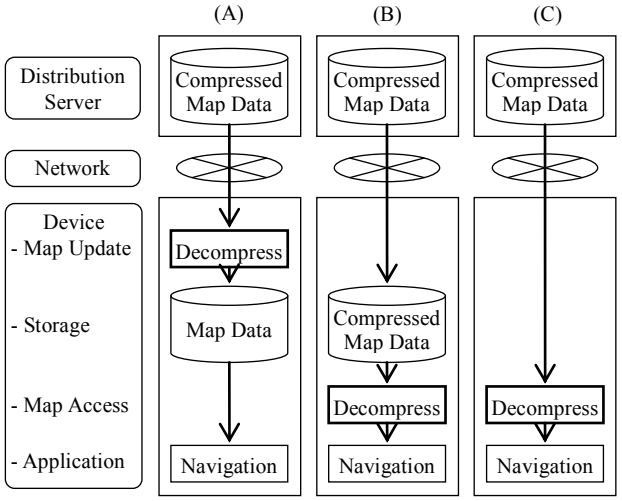

図 2　地図データ圧縮の適用パターン
Figure 2　Patterns of the application of map compression.

図に記載した分類のうち, (A)は地図データを更新する処理の過程で圧縮データを伸張する方式である．この場合，データ圧縮の主な目的は通信サイズの削減であり，高い圧縮率が求められる．また，データ圧縮に加え，地図データの差分のみを配信することでサイズを削減する手法も用いられる[10]．(B)はストレージに地図データを圧縮したまま格納し，アプリケーション実行中に逐次データを伸張する方式である．この場合，圧縮データの伸張処理に伴うナビ性能の低下をユーザが感じないようにすることが重要であり，高い伸張性能が求められる．(C)はアプリケーション実行時に直接配信サーバーから地図データを取得する方式であり，通信環境の影響を大きく受けるため，適用する地図データの範囲等，データ圧縮以外の検討も必要である．

上記の分類のうち, 従来はナビ性能を確保するために(A)を採用することが多いが，常に最新のデータを利用可能な(C)の実現を視野に，本稿ではまず(B)を対象として検討する．そのためナビ性能の低下をユーザが感じないように圧縮データを高速に伸張可能とすることを優先し，その上で可能な限り圧縮率を高めることを図る．

3.2 課題

上記の要件に対して，関連研究を踏まえ，本稿で解決する課題を以下に記載する．

(1) 列指向圧縮データを伸張する際のキャッシュ効率

RDB における列指向データ圧縮の基本的な考えは，特定の列のみを参照するクエリが多いため，列毎に圧縮を行う

ことで性能を向上可能な点である．一方，カーナビ用の地図データの場合，ナビ性能を確保するためにデータベースを設計する時点で機能毎にテーブルが分割され，各機能はレコードの全ての列を参照することが多い[11][12]．また，同時に参照されるレコードがなるべく近い位置に格納されるような工夫もなされる．そのためクラスタのような一定のまとまりでリードするほうが性能が良く，圧縮データの伸張も全ての列を伸張する処理が基本となる．

全ての列を伸張する場合，2.2 節で述べた PAX のようにキャッシュ上にデータを配置することで伸張処理の高速化を図れる．しかし PAX はデータ圧縮を想定したモデルではないため，入出力データ（圧縮前後のデータ）や復号用のワーク領域を考慮したデータ配置を考える必要がある．

(2) 軽量符号の組合せによる性能向上

前述した関連研究では，複数の符号化方式を組合せ，列毎の特性に応じて適切な方式を選択することで性能向上を図っている．本研究も基本的に同様のアプローチとするが，あらゆるデータに対して伸張性能と圧縮率を両立可能な方式は存在しないという問題がある．

例えば Abadi 等による取組みでは，ランレングス符号等の軽量符号以外，例えばハフマン符号[13]は RDB の性能を維持する上で採用困難だったとしている．また，郡等は，RID 符号は高い圧縮効果を示すが，数値データ等，十分な効果を得られない一部のデータが圧縮後データサイズの大半を占める問題が残るという考察を示している．

この問題に対し，本稿ではユーザが性能低下を感じないことを目標にし，圧縮対象のテーブルを参照する各機能の性能要件に応じて，使用する符号化方式を選択する方式を考える．例えばナビ画面に表示する情報を格納しているテーブルの場合，多少復号性能が悪くても，画面の表示に必要な 1 回のリードサイズが十分に小さくリード時間の増加が数百ミリ秒程度に収まれば，ユーザは性能低下を感じないと考えられる．本稿ではハフマン符号等の低速な符号も含めてより多くの軽量な符号化方式を採用し，圧縮対象のテーブルに応じて使用する方式を選択することにより，伸張性能と圧縮率を高めることを検討する．

3.3 目標

以上を踏まえ，本研究の目標を記載する．ストレージからのデータリード性能を d_s [MB/sec]，圧縮前のデータサイズを s_u [MB]，圧縮後のデータサイズを s_c [MB]，圧縮率 r を s_c / s_u とする．また，圧縮データの伸張時間を t [sec]としたとき，伸張性能 d_c を s_u / t [MB/sec]とする．この時，データを圧縮しない場合のリード時間は以下になる．

$$t_u = \frac{s_u}{d_s} \quad (1)$$

一方，データを圧縮した場合のリード時間は，伸張時間を含めると以下になる．

$$t_c = r\frac{s_u}{d_s} + \frac{s_u}{d_c} \quad (2)$$

ユーザにナビ性能の低下を感じさせないため，t_c が数百ミリ秒を超えるような場合は t_u と t_c を同程度にしつつ，一般的な圧縮方式と同等以上の圧縮率 r の達成を目標とする．なお，一般的な圧縮方式として，OSS の汎用圧縮ライブラリである Zlib（deflate）との比較を行う[14]．

4. 提案方式

4.1 概要

本章では，提案方式の詳細を述べる．まず，提案方式による圧縮データの構造を図 3 に示す．図の左側は提案方式が対象とする入力データ（圧縮前のデータ）の構造と圧縮処理の流れを示しており，図の右側は圧縮後のデータ構造を示している．提案方式は，固定長のレコード（1 レコードのサイズを N バイトとする）が並んだデータを対象とするものであり，入力データ全体を複数のブロックに分割してブロック毎に圧縮を行う．各ブロックは M 個のレコードを含んでおり，M の値はシステムが使用する SoC のキャッシュ仕様を考慮して決定する（4.2 節で詳細を述べる）．

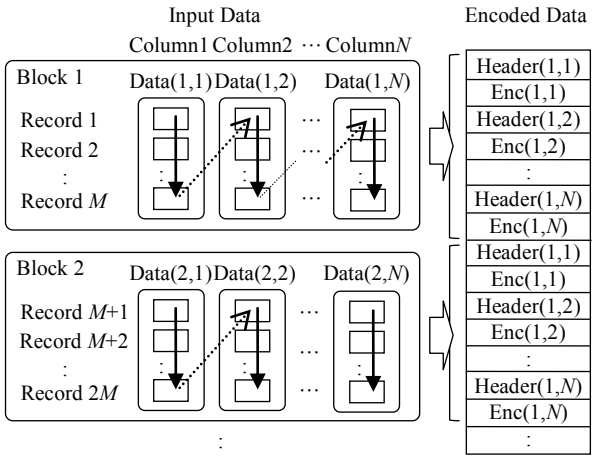

図 3　提案方式のデータ構造
Figure 3　Data structure of proposal method.

各ブロック内では，レコードを構成する各バイトデータを列として考え，列毎に符号化を行う．例えばブロック 1 の列 1 の符号化では，各レコードの 1 バイト目の値を，レコード 1 から M の順に参照して符号化する．図では，この入力データを Data(ブロック番号,列番号)，これを符号化したデータを Enc(ブロック番号,列番号)と表記している．この符号化を行う際に使用する方式は，複数の符号化方式の中から，圧縮後のサイズが最も小さくなる方式を選択して行う（4.3 節で詳細を述べる）．選択された符号化方式は列毎のヘッダ領域（Header）に格納する．各ヘッダは 1 バイトであり，そのうち 4 ビットを符号化方式の識別子として，残り 4 ビットを方式毎のパラメータとして使用する．

復号処理は上記と逆の手順となる．すなわち，圧縮後のデータに対し，ヘッダ 1 バイトをリードして符号化方式を特定し，ヘッダ以降に続く圧縮データから M バイトを復号する処理を繰り返す．

4.2 伸張時のキャッシュレイアウト

上記手順のうち，各ブロックに含まれるレコード数 M の値について述べる．列指向で圧縮されたデータの伸張を高速に行うため，3.2 節(1)で述べたように，PAX を参考にして入出力データや復号用ワーク領域を考慮したキャッシュのデータ配置を考える．以降，伸張処理を例にして記載するが，圧縮処理の場合も基本的に同様である．

提案方式は，近年の一般的な方式であるセットアソシアティブによるキャッシュメモリを対象とする．表 1 にセットアソシアティブ方式のパラメータを示す．なお，以降ではウェイ数 w を 4 以上とし，このうち入力データが使用するウェイ数を w_s，出力データが使用するウェイ数を w_d，復号用ワーク領域が使用するウェイ数を w_t とする．

表 1　セットアソシアティブ方式パラメータ
Table 1　Parameters of set-associative.

項目	パラメータ
ラインサイズ	2^b [byte] (オフセットサイズ: b [bit])
エントリ数	2^a (インデックスサイズ: a [bit])
ウェイ数	w (≧4)
置換方式	LRU または FIFO

表 1 に示すパラメータの場合，サイズ 2^{a+b} ごとのメモリ空間で同じエントリを利用できるのは w 個である．そのため $w\times2^{a+b}$ より広い範囲で等間隔にメモリを参照する処理を繰り返すとキャッシュミスヒットが発生し始め，$2w\times2^{a+b}$ を超えるとキャッシュを完全に外すことになる．

一方，圧縮データを列毎に分割されたデータに復号後，これをレコード順に並んだデータに復元する処理は，基本的に以下となる．ここで配列 *src* および *dest* はそれぞれ列毎のデータ（入力データ），およびレコード順に並んだデータ（出力データ）の格納領域であり，変数 t および u の初期値は 0 である．

```
for i := 1 to N step 1 do
    u := i;
    for j := 1 to M step 1 do
        dest[u] := src[t];
        t := t + 1; u := u + N;
```

上記の処理はサイズ $M\times N$ のメモリ空間において等間隔でメモリを参照するため，キャッシュミスヒットを回避するためには式(3)が成立する必要がある．

$$M \cdot N \le w_d \cdot 2^{a+b} \tag{3}$$

次に式(3)における w_d について考察する．上記処理の内側ループにおいて出力データ格納領域に相当する全てのラインが参照される一方，入力データは 1 バイトずつリードされ，リード済の入力データを格納しているラインはその後は参照されない．よって置換方式が LRU の場合は，新たな入力データを DRAM からリードする際に置き換わるラインは既にリード済の入力データであり，w_s は基本的に 1 になる．一方，置換方式が FIFO の場合，出力データ格納領域の参照アドレスが等間隔になっており使用エントリが分散されるため，出力データ格納領域が置き換え対象になることが多い．そのため w_d ≦ 1 となるように M を設定する必要がある．以上から，式(3)はキャッシュの置換方式に応じて式(4)または式(5)になる．

$$M \le \frac{1}{N}(w-1-w_t)2^{a+b} \qquad \text{(LRU の場合)} \tag{4}$$

$$M \le \frac{1}{N}2^{a+b} \qquad \text{(FIFO の場合)} \tag{5}$$

上記のように M を制限した場合における，キャッシュ上のデータ配置を図 4 に示す．図は 4 ウェイセットアソシアティブ（ラインサイズ 16 バイト，256 エントリ）のキャッシュメモリで，1 レコード 16 バイトのデータを処理する場合の例である．提案方式では，ワーク領域（work area）が 1 ウェイ相当の空間を超えない符号化方式のみを利用することとし，1 ウェイを入力データのリード用（src），残りを出力データのライト用（dest）と考える．

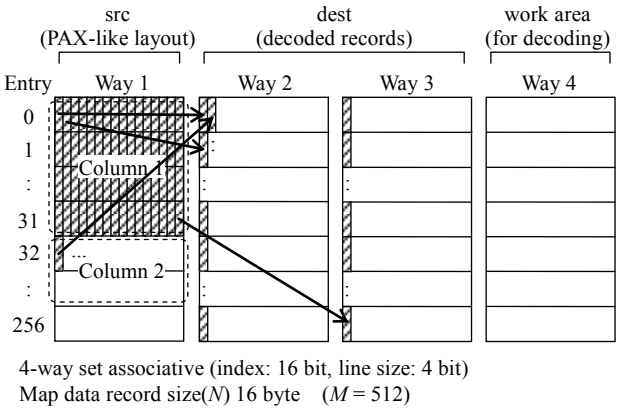

図 4　提案方式におけるキャッシュ上のデータ配置
Figure 4　Cache data layout of proposal method.

4.3 軽量符号化方式の組合せ

次に，複数の符号化方式を用いて各列のデータを符号化する処理について述べる．提案方式では，ワーク領域が 1 ウェイ相当の空間を越えない軽量な符号化方式のみを使用するものとし，その上で，圧縮対象のデータに必要な伸張性能に応じてあらかじめ選択した複数の符号化方式のうち，圧縮後のサイズが最も小さくなる符号化方式を選択する．なお，いずれの方式も圧縮前よりサイズが小さくならない場合，圧縮せずにデータを格納する．

表 2 は，本稿で想定する列毎の特性と，それぞれの特性に対して効果的と考えられる符号化方式を記載したものである．列の特性は，同一記号の連続性，出現する記号数，記号の出現頻度に関する確率分布の有無，および頻出記号の有無の 4 つの観点で分類している．各符号化方式の概要は以下である(括弧内は以降で用いる各符号の表記である)．

(1) ランレングス符号（RLE）

同一記号の繰り返しを記号と長さの組で置換する．

(2) 固定長符号（FF）

各記号に固定長の符号を割り当てる．出現する記号の数を x としたとき，1 つの記号の符号長は ceil(log x)となる．本方式は x が 16 以下の場合のみ有効となる．

(3) アルファ符号（unary，単進符号）（A）

記号が意味する整数値を「0」の個数で表現し，区切りに「1」を出力する．例えば「123」は「010010001」となる．記号の出現頻度に小さい数値を中心にした確率分布がある場合に有効である．

(4) ライス符号[15]（RICE）

記号が意味する整数値を 2^k で割った商 p と剰余 q を求め，p のアルファ符号と q の下位 k ビットを出力する．例えば k=2 の場合の「4」は，p のアルファ符号「01」と q の下位 k ビット「00」を結合した「0100」になる．アルファ符号と同様に出現頻度に小さい数値を中心にした確率分布がある場合に有効であり，k を大きくすると分布が緩やかになる．提案方式では列毎に k を 1 から 8 の範囲で変化させ，圧縮後のサイズが最も小さくなる k を採用する．

(5) エスケープ符号（ESC）

最頻出記号を「0」で符号化し，それ以外の記号はエスケープとして「1」を出力し，続けて記号の 8 ビット値をそのまま出力する．特定の記号が頻出し，かつ他の記号も数回出現するような列用に用意する．

(6) Canonical Huffman Code[16]（CHC）

特に特性が無い場合はハフマン符号（最小冗長符号）を用いる．なお，通常のハフマン符号はハフマン木用のワーク領域が必要になるため，ハフマン木を用いずに表引きのみで処理可能な Canonical Huffman Code を用いる．

表 2　列の特性と対応する符号化方式
Table 2　Column characteristics and coding algorithms.

連続性	記号数	分布	頻出	対応方式
あり	－	－	－	ランレングス符号
なし	少ない	－	－	固定長符号
	多い	あり	－	アルファ/ライス符号
		なし	あり	エスケープ符号
			なし	最小冗長符号（CHC）

また，上記の符号化方式に対して，以下の処理を組み合わせて符号化を行う．

(7) 差分符号

記号を数値と考え，隣接する記号間の差分を符号化する．差分を計算することで表 2 に記載した特性が現れる列に有効と考えられる．

(8) 記号表

記号の一覧（記号表）を出現頻度順に出力した上で表内のインデックスを符号化する．圧縮データに記号表を含むため，出現記号の数が少ない場合に有効と考えられる．

表 3 に提案方式で使用する符号化の一覧を記載する．表において「○」は提案方式で使用することを示し，「－」はその組合せは無いことを示している．例えば固定長符号（FF）と CHC は記号表を必ず出力する．「×」は，組合せとしてはあり得るが，本稿で評価の対象としたデータでは大きな効果が無かったため，コードサイズを削減するため使用していないことを意味する．

表 3　提案方式で使用する符号化方式
Table 3　Coding algorithms used by proposal method.

符号化方式	組合せ方式 (表記)		
	なし	差分符号	記号表
ランレングス符号	○(RLE1)	×	×
固定長符号	－	×	○(FF3)
エスケープ符号	○(ESC1)	○(ESC2)	○(ESC3)
アルファ符号	×	○(A2)	○(A3)
ライス符号	×	×	○(RICE3)
CHC	－	×	○(CHC3)
非圧縮	○(COPY)	－	－

5. 評価

提案手法をカーナビに実装し，地図データに含まれる 2 種類のデータに対して性能評価を行った．表 4 に評価に用いたシステムの仕様を記載する．

表 4　評価環境のキャッシュメモリ仕様
Table 4　Cache memory spec. of target platform.

項目		仕様
SoC		ルネサス社製 SH4
キャッシュメモリ	方式	4-way set associative
	ラインサイズ	32byte（オフセット 5bit）
	エントリ数	256（インデックス 8bit）
	置換方式	LRU

表 5 に評価に使用したデータを記載する．データ A は音声認識用のデータであり，1 度にリードするサイズが最大で数十 MB になるため，データを圧縮しない場合のリード時間 t_u とデータを圧縮した場合のリード時間 t_c をほぼ同程度にすることが求められる．データ B は目的地検索用の施設データであり，1 度にリードするサイズは高々数十 KB であるため，t_c が数百ミリ秒程度に収まれば良い．

ⓒ2016 Information Processing Society of Japan

また，予備評価として，評価環境のストレージから上記データをリードした時間（t_u）を測定した結果を表 6 に示す．なおストレージは HDD であり，HDD の設定やリード処理の入出力単位等は以降の評価でも同様である．

表 5　評価用テストデータ
Table 5　Test data for performance evaluation.

データ	サイズ(S_u) [byte]	N [byte]	リード単位 [byte]	備考
A	147,931,135	16	数十 MB	音声認識データ
B	42,916,896	24	数十 KB	施設データ

表 6　ストレージリード時間（t_u）
Table 6　Storage read time (t_u).

データ	リード時間(t_u) [msec]	リード性能(d_s) [MB/sec]
A	6,073	23.23
B	1,670	24.50

5.1 圧縮データリード性能

まず，提案方式による圧縮率と伸張性能，および評価環境における圧縮データリード性能の評価結果を示す．評価に際し，データ A は高い伸張性能が求められるため RLE1 と FF3 のみを用いて圧縮を行った．一方，データ B はデータ A ほどの伸張性能は求められないため，表 3 に記載した全ての符号化方式を用いて圧縮を行った．また，いずれの場合も復号用ワーク領域 $w_t = 1$ とし，ブロック内のレコード数 M は後述する評価結果を踏まえて 1024 とした．

表 7 と図 5 に圧縮率と伸張性能の評価結果を記載する．データ A は，Zlib とほぼ同様の圧縮率で約 8 倍の伸張性能になった．データ B は圧縮率と伸張性能のいずれも Zlib を上回ったが伸張性能は Zlib の約 2 倍に留まった．

表 7　圧縮サイズと伸張時間
Table 7　Compressed size and decompression time.

データ	提案方式		Zlib	
	圧縮サイズ [byte]	伸張時間 [msec]	圧縮サイズ [byte]	伸張時間 [msec]
A	68,006,979	2,460	60,725,032	20,063
B	7,257,963	1,920	9,968,208	3,970

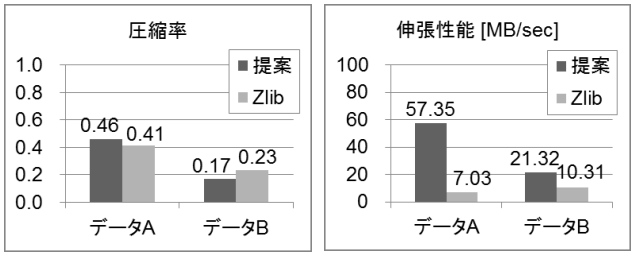

図 5　圧縮率と伸張性能
Figure 5　Compression ratio and decompression throughput.

表 8 は，それぞれの方式による圧縮データをリードした時間（t_c）を測定した結果である．この時間にはリード処理に伴う各種のオーバーヘッドも含まれている．この結果と，非圧縮時のリード性能（表 6）を比較した結果を図 6 に示す．データ A について，提案方式は非圧縮時とほぼ同等の性能になった．データ B については，提案方式でも非圧縮時の性能に及んでいないが，1 回のリードサイズが数十 KB 程度であるため画面表示に要する時間の増加は数ミリ秒程度であり，ナビ性能の低下を感じるほどではないと思われる．以上の結果から，データ A と B については提案方式は 3.3 節の目標を達成していると言える．

表 8　圧縮データリード時間（t_c）
Table 8　Compressed data read time.

データ	提案方式		Zlib	
	リード時間 (t_c) [msec]	リード性能 [MB/sec]	リード時間 (t_c) [byte]	リード性能 [MB/sec]
A	6,330	22.29	24,323	5.80
B	2,430	16.84	4,870	8.40

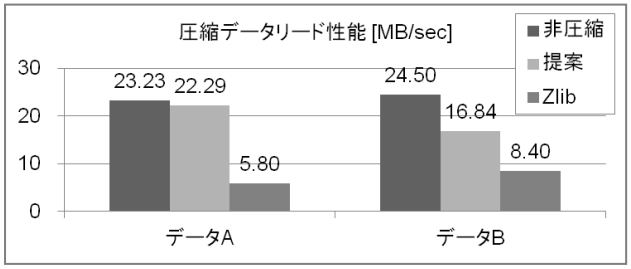

図 6　圧縮データリード性能の比較
Figure 6　Comparison of compressed data throughput.

5.2 キャッシュレイアウトの効果

以降，本稿で述べた手法の効果を確認する．まず，4.2 節の検討結果を確認するため，データ A に対して RLE1 のみを用いて圧縮を行い，ブロック内のレコード数 M と，伸張性能の関係を確認した．なお，RLE1 はワーク領域をほとんど必要としないため $w_t = 0$ として考える．表 4 に示したキャッシュメモリを用いて 1 レコード 16 バイトのデータを圧縮する際の M の上限は，式(4)から 1536 である．この値の近辺における伸張性能の変化を確認するため M = 896 から 2048 の範囲で測定を実施した．また，レコード数上限を考慮しない場合の性能として，キャッシュを完全に外す $M = 4096$ の場合を測定した．表 9 と図 7 に結果を示す．

図 7 が示すように，レコード数 M の増加に応じて伸張性能が低下し，特に $M = 1536$ を超えた時点から性能低下が著しくなる．$M = 4096$ の場合は，$M = 1536$ の場合と比較して約 4.6 倍の伸張時間になった．なお，$M = 1536$ 未満の範囲においても M に応じて性能が低下している．伸張以外の処理によってキャッシュの置き換えが発生しているためと

思われるため，M は式(4)を上限として，圧縮率に影響を与えない範囲で可能な限り小さくしたほうが良いと言える．

表 9　レコード数 M と性能測定結果

Table 9　Record number M and performance.

M	伸長時間[msec]	圧縮後サイズ[byte]
896	6,353	83,520,108
1024	6,540	83,478,748
1152	6,643	83,497,460
1280	6,633	83,479,296
1408	6,736	83,466,156
1536	6,783	83,457,816
1664	7,166	83,465,340
1792	7,320	83,457,740
1920	7,903	83,464,940
2048	8,276	83,435,688
:	:	:
4096	31,356	83,127,464

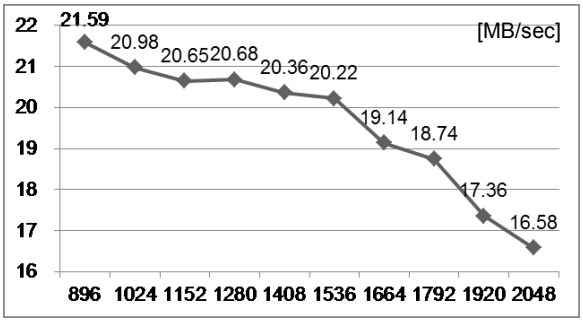

図 7　レコード数 M と伸張性能

Figure 7　Record number M and decompression throughput.

5.3 軽量符号化方式の組合せの効果

次に 4.3 節の検討結果について確認する．表 10 は，データ A を処理する際に各符号化方式で処理したサイズ，圧縮後のサイズ，および伸張時間を測定した結果である．なお，処理したサイズが大きいほど，その方式が圧縮率の向上に有効であったことを意味する．例えば表 10 の 1 行目は，約 51 MB のデータに対して RLE1 が選択されて圧縮サイズが約 25 MB になり，伸張に 973 ミリ秒を要したことを意味する．RLE1 と FF3 の両方とも高い伸張性能を示しているが，圧縮効果が無かったデータ（COPY）が多くあり，圧縮率が Zlib と同程度に留まった要因になっている．また FF3 の圧縮率が高く，圧縮データのリード時間が短縮されたことが伸張性能向上に寄与している．つまり提案方式の性能は各列で出現する記号数に大きく依存する．

表 11 は，データ B の場合の測定結果である．RLE1 および FF3 とそれ以外の方式で伸張性能に大きな差がある．この理由は，RLE1 と FF3 はバイト単位のデータ処理で実装しているのに対し，その他の方式はビット単位のデータ処理を行っているためと思われる．

表 10　データ A に対する測定結果

Table 10　Performance for data A.

方式	処理サイズ [byte]	圧縮サイズ [byte]	伸張時間 [msec]
RLE1	51,955,712	25,219,144	973
FF3	64,608,256	11,420,667	1076
COPY	31,367,168	31,367,168	410

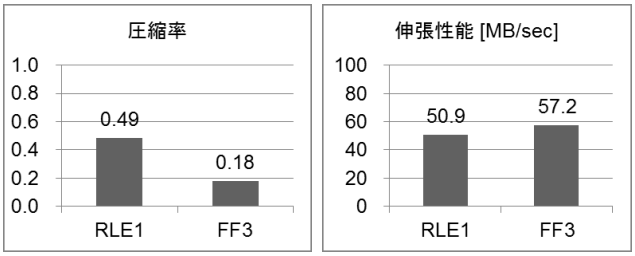

図 8　各符号化方式の性能比較（データ A）

Figure 8　Comparison of each coding (Data A).

表 11　データ B に対する測定結果

Table 11　Performance for data B.

方式	処理サイズ [byte]	圧縮サイズ [byte]	伸張時間 [msec]
RLE1	1,185,998	153,301	30
FF3	27,367,978	79,018	390
ESC1	51,150	43,142	10
ESC2	3,862,166	2,576,437	333
ESC3	1,829,806	388,560	110
A1	929,566	235,138	146
A2	2,671,394	547,213	240
RICE3	1,376,958	1,219,984	243
CHC	2,758,690	1,131,980	350
COPY	883,190	883,190	16

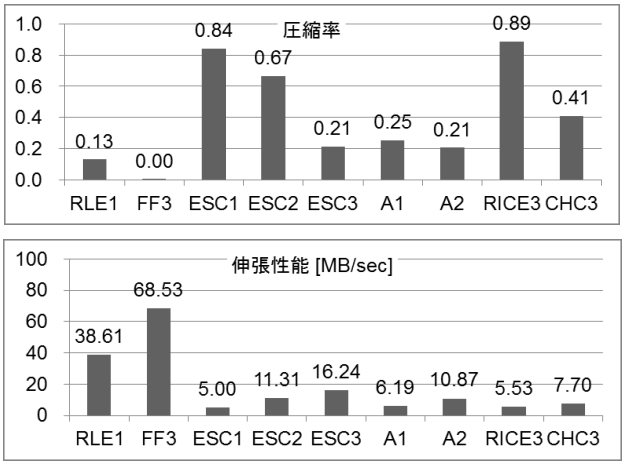

図 9　各符号化方式の性能比較（データ B）

Figure 9　Comparison of each coding (Data B).

以上の結果から，RLE1 および FF3 では圧縮効果が無いデータに対して，本稿で評価した符号化方式は圧縮率の観点では有効だが伸張性能は十分とは言えない．データ A お

よび B については目標を達成しているが，より多様なデータに対応するためには，高速かつワーク領域が 1 ウェイに収まる別の符号化方式が必要になる．

6. まとめ

本稿では，地図更新を容易化するための取組みの一環として，カーナビのストレージに格納している地図データの圧縮方式について述べた．圧縮データの伸張処理によるナビ性能低下をユーザが感じないよう，列指向データ圧縮をベースにキャッシュの効率化と複数の軽量な符号化方式を利用することで，Zlib 相当の圧縮率と非圧縮時同等のリード性能の実現を図った．提案方式を実装して評価した結果，特定のデータに対する上記性能の実現性を確認した．

今後，カーナビはスマートフォンや通信ユニットを介してネットワークに常時接続する動きが進むと考えられる．データ更新に必要なストレージ容量や通信サイズを削減しつつ，地図自動更新等，常時接続環境を活かしたデータ更新作業の容易化に取り組んでいく予定である．

参考文献

[1] B. R. Iyer and D. Wilhite, Data Compression Support in Databases, In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), pp. 695-704 (1994).
[2] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik, C-Store: A column-oriented DBMS, In Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), pp. 553-564 (2005).
[3] D. J. Abadi, S. R. Madden, and M. C. Ferreira, Integrating Compression and Execution in Column-oriented Database Systems, In Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, pp. 671-682 (2006)
[4] J. Ziv and A. Lempel., A Universal Algorithm for Sequential Data Compression, IEEE Transactions on Information Theory, Vol. 23(3), pp. 337-343 (1977)
[5] 郡光則, データウェアハウス向け高性能データ圧縮方式, 情報処理学会論文誌, Vol. 47, No. SIG13, pp. 58-73 (2006)
[6] A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis, Weaving Relations for Cache Performance, In Proceedings of the 27th International Conference on Very Large Data Bases (VLDB), pp. 169-180 (2001)
[7] H. Zhang, G. Chen, B. C. Ooi, K. Tan, M. Zhang, In-Memory Big Data Management and Processing: A Survey, IEEE Transactions on Knowledge and Data Engineering, pp. 1920-1948 (2015)
[8] R. Ramakrishnan and J. Gehrke, Database Management Systems, McGraw-Hill, 2 edition (2000)
[9] G. P. Copeland and S. F. Khoshafian, A Decomposition Storage Model, In Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 268-279 (1985).
[10] 淺原彰規, 谷崎正明, 嶋田茂, 森岡道雄, 道路の接続性を保障したテレマティクスサービスのための地図差分更新方式, 情報処理学会論文誌, Vol. 49(1), pp. 221-232 (2008).
[11] 臼井真人, 福山薫, カーナビゲーション用フォーマット・KIWI フォーマットのデータ更新技術, 地理情報システム学会講演論文集, Vol. 16, pp. 227-230 (2007).
[12] Navigation Data Standard (NDS), http://www.nds-association.org/
[13] D. Huffman, A Method for the Construction of Minimum-Redundancy Codes, Proceedings of the Institute of Radio Engineers (IRE), Vol. 40(9), pp. 1098-1101 (1962).
[14] RFC1950 ZLIB Compressed Data Format Specification version 3.3, IETF
[15] R. F. Rice and J. R. Plaunt, Adaptive Variable-Length Coding for Efficient Compression of Spacecraft Television Data, IEEE Transactions on Communications, Vol. 16(9), pp. 889–897 (1971).
[16] I. H. Witten, A. Moffat, and T. C. Bell, Managing Gigabytes: Compressing and Indexing Documents and Images, Morgan Kaufmann (1999).

ⓒ2016 Information Processing Society of Japan