情報処理学会論文誌 Vol. 50 No. 2 777–786 (Feb. 2009)

辞書データ送付による携帯端末S/W更新時のBPE高速化方式

清原 良三[†1,†2] 三井 聡[†2]
沼尾 正行[†3] 栗原 聡[†3,†4]

近年，携帯電話を代表とする携帯端末のソフトウェア規模が増大し，多数の機器で出荷後に不具合の修正を必要としている．そのため，出荷後にソフトウェア更新機能は必須となっている．携帯端末は，コストの面からNAND型フラッシュメモリに圧縮状態でプログラムコードを置き，デマンドページング方式で実行する方式が採用されつつある．デマンドページングに適した圧縮方式であるBPE（Byte Pair Encoding）方式は圧縮に時間がかかるという欠点がある．本論文では，携帯端末のソフトウェアを更新する際に，端末上で既存のBPE方式，圧縮率の変更をともなわない高速なプログラムコード圧縮手法を提案し，実際の出荷ソフトウェアイメージを利用して評価し，効果を示す．

Accelerating Technique of BPE with Downloaded Dictionaries for S/W Updating on Mobile Devices

Ryozo Kiyohara,[†1,†2] Satoshi Mii,[†2]
Masayuki Numao[†3] and Satoshi Kurihara[†3,†4]

Recently, the software size of embedded systems is increasing rapidly. The software of mobile phones is a typical example; its complicated processing such as event handling causes the need for bug fix after shipment. On the other hand, NAND Flash memory devices are adopted in mobile phones in order to reduce the cost. Compressed program codes are stored on NAND flash memories. The program codes are loaded and extracted to RAM using demand paging technologies. In this case, the program codes need to be compressed on mobile phone in software updating function which needs fast compression algorithm. BPE (Byte Pair Encoding) method is suitable for these architecture. In this paper, we propose a fast BPE algorithm on mobile phones for software updating. Our evaluation on mobile phones shows that the proposed method is effective to compress the program codes.

1. はじめに

最近の携帯端末では，コストを削減するために実行プログラムコードをNAND型フラッシュメモリに格納したうえで，実行時にプログラムコードをRAMにコピーして実行する形式を採用しつつある．コスト削減が目的であるため，NOR型フラッシュメモリでXIP[⋆1]実行し，ほとんどのプログラムコードをRAMにコピーして実行しない場合に比較してRAM容量を増加させるわけにはいかない．そこでRAM容量不足に対応するためデマンドページング方式を採用する傾向にある．

また，NAND型フラッシュメモリの節約とロード時間短縮のためにNAND型フラッシュメモリ上では圧縮した形式でプログラムコードを配置する．圧縮の展開時に時間のかかるような圧縮方式では性能に大きく影響があるため，展開が高速にできる圧縮方式を採用することが重要となる．BPE（Byte Pair Encoding）方式は圧縮率でも他の圧縮方式にそれほど劣らずに，展開時間が短い方式であり，こういったデマンドページング方式に適用するには最適な圧縮方式の1つである．しかしながら，圧縮には何度もデータを走査する必要があり圧縮速度は非常に遅いとされている．

通常の携帯端末の開発では，高速なCPUを持つクロス環境上で開発し，実行イメージを作成したうえで携帯端末にダウンロードするためこの遅さは問題にならない．また，ユーザの手元では，展開するのみなので圧縮速度は関係がない．しかし，最近の携帯端末で必須となっている出荷後にネットワークを経由してソフトウェアをダウンロードし，ソフトウェアを書き換えるというソフトウェア更新に対して重大な問題となる．ソフトウェア更新では，ダウンロード時間を短くするための差分抽出技術や，バグFIXをしても差分が出にくいソフトウェア構成とするためのソフトウェア構成法が重要であるが，さらに最近のようにソフトウェアが大規模化すると，差分適用時間を短くすることが重要である．

†1 大阪大学大学院情報科学研究科
Graduate School of Informartion Science and Technology, Osaka University
†2 三菱電機株式会社情報技術総合研究所
MITSUBISHI ELECRIC CORPOLATION INFORMATION TECHNOLOGY R & D CENTER
†3 大阪大学産業科学研究所
The Institute of Scientific and Industrial Research, Osaka University
†4 独立行政法人科学技術振興機構，CREST
Japan Science and Technology Agency, CREST
⋆1 eXecute In Place

圧縮形式でデータを新版と旧版で比較すると差分は非常に大きくなる傾向にある．そのため，非圧縮状態で差分を抽出し，展開状態に差分を適用し，再度携帯端末上で圧縮してデータを書き込む必要がある．各段階での実行の高速性が求められ，圧縮時間の遅さは致命的な問題となる．

本論文では，クロス開発環境上で圧縮状態を作成したときの圧縮辞書情報を抽出し，この抽出した情報を端末上での圧縮用辞書として利用することにより，携帯端末上で高速に圧縮する手法を提案するとともに，トレードオフとなる差分データ量を抑える手法を提案，評価する．

2. 関 連 研 究

携帯端末のソフトウェアの更新に関しては各キャリアが店頭で書き換えサービスを実施したり，ネットワークを経由してダウンロードしユーザの手元で書き換えたりするサービスなどがある[1),2)]．これらを支える技術は，ソフトウェアの構成法[3),4)] や，バイナリ差分抽出の技術[5)–8)] である．しかし，これらは NOR 型のフラッシュメモリ上で XIP で実行しているプログラムコードを想定したものであり，圧縮したプログラムコードに対するソフトウェアの更新という観点では検討されていない．

しかし，携帯端末向けでは，NAND 型フラッシュメモリ上にプログラムコードを置き，デマンドページング方式を採用した OS も登場しており[9)]，効率的に NAND 型フラッシュメモリを利用するには圧縮プログラムコードに対応したソフトウェア更新技術が必要となると考えられる．

また，圧縮方式においては，古くから様々な研究開発が行われ，実用化されているものは多い．圧縮には多少時間がかかっても，圧縮率がより高く，展開がそれなりに速ければ良いという場合が少なくないため[10)]，圧縮率の高さと高速な展開方式の実現に関しての研究が多く行われている[11)]．また，実行プログラムコードの圧縮という観点からも様々な方式が研究開発され，多くの方式が提案されている[12)]．

NAND 型フラッシュメモリ上に圧縮プログラムコードを置き，デマンドページング方式をとる実行形態に適した圧縮方式の 1 つとして BPE 方式[13)] の利用が提案され，効果的であることが示されている[14)]．BPE 方式では必要な部分だけの展開も可能であり，ソフトウェアの更新を考慮しなければこの手法は非常に有効な手法である．しかし，BPE 方式は圧縮に時間がかかるという課題がある．

携帯端末での実行性能とソフトウエア更新時における展開や圧縮の高速化の必要性を想定し，携帯端末でよく使われる CPU アーキテクチャに依存してはいるものの，効率の良い圧縮方式に関する研究開発も行われている[15)]．

文献 16) では BPE に対して部分展開可能であることから文字列の比較の研究を行っている．文献 17) は圧縮データを展開せずに文字列比較をするという観点から，BPE の圧縮処理高速化に関する研究を行っている．この研究では BPE の圧縮速度の遅さを 1 回の走査で最も多いペアだけを発見して使っていない 1 バイトに置き換える処理そのものにあると考え，複数のペアを選択して同時に置き換えていくことで高速化を図っている．しかしながら，圧縮率が少し落ちるという問題が発生する．圧縮率の低下は本論文で対象とするデマンドページング方式に対してはページの読み込み時の読み込み時間に影響するためできるだけ避けるべきである．

そこで，本論文ではソフトウェア開発の効率化という観点からも既存の BPE 方式の仕様を変更せず，すなわち展開ロジックなどの変更をせずに，また圧縮率を変えずに高速圧縮する手法に関して述べ，実際の携帯電話ソフトウェアイメージで評価し効果を示す．

3. ソフトウェア更新と BPE

3.1 携帯端末とコード圧縮

携帯端末上では，**図 1** に示すように，NAND 型フラッシュメモリ上にプログラムコードを RAM 上の非圧縮状態におけるページに相当する単位ごとに圧縮して保持する．RAM 上で実行時にページフォルトを起こすと，該当ページの圧縮状態のプログラムコードを展開して RAM 上にロードする．圧縮して保持するメリットは，NAND 型フラッシュメモリ領域の節約とともに，読み込みデータ量の削減による読み込み処理の高速化もあげられる．

読み込みの性能的には圧縮してあると NAND 型フラッシュメモリを読む時間は短くなるが，展開する時間が余分に必要になる．そこで，高速に展開可能な圧縮アルゴリズムを採用する必要がある．

本論文ではこのような圧縮に適したアルゴリズムである BPE 方式[13)] を対象とする．

3.2 ソフトウェア更新方式

携帯端末などの大規模なソフトウェアを更新する際の一般的な手順は，文献 1)，2)，6) に示されるように，以下の手順となる．

(1)　開発環境上で新しい版のソフトウェアイメージを作成する．
(2)　旧版と新版との間でバイナリ差分を抽出する．
(3)　差分情報をネットワークなどを利用して端末に転送する．

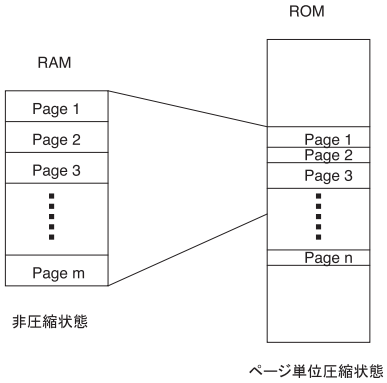

図 1　圧縮 ROM イメージ
Fig. 1　ROM image of compressed code.

(4)　端末をソフトウェアの書き換えモードに変更する．
(5)　旧版のイメージにバイナリ差分を適用して新版のイメージに変更する．
(6)　端末をリセットする．

このようなソフトウェア更新におけるユーザからみた機能要件は以下である．

(1)　ソフトウェアの書き換え時間を短くすること．
(2)　ダウンロード時間を短くすること．
(3)　安全に書き換えられること．

ソフトウェアの書き換え処理では，フラッシュメモリを書き換える動作を行う．フラッシュメモリはコスト面から二重化しているわけでもないため，実行しているコードを書き換えることになる．NAND 型フラッシュメモリでは RAM にロードして実行を行うため，実行中の書き換えが可能ではあるが，デマンドページングなどを採用していると，フラッシュメモリへのアクセスがいつ行われるか分からないため，実質的にはソフトウエア実行中のフラッシュメモリの更新は単純には不可能になり，書き換え処理中は携帯電話のプログラムを停止する必要がある．

そのため，ソフトウェアの書き換え中は携帯電話の基本的なサービスである電話の発着信やメールの送受信を行うことができなくなるため，この書き換え時間を短くする必要がある．ソフトウェアの書き換え時間を短くするためには，フラッシュメモリの書き換えるブロック数を抑えることが重要となる[4]．

図 2　BPE 方式
Fig. 2　BPE.

一方，ダウンロード時間はデータ転送量で決まる．データのダウンロード中も携帯電話の基本的なサービスである発着信やメールの送受信を行うことができるため，ユーザビリティをそれほどは失わないものの，RAM を大量に使うことは明らかで，メモリを大量に消費するであろう一部のアプリケーションの動作が制限されることになる．また，不具合によるソフトウェアの更新ではサービスの利用者が集中することが想定され，携帯電話網への影響も考慮する必要がある．

よって，データ転送量を抑えておくことが望ましい．このために新版と旧版の差分データを転送することが多い[5]．

3.3　BPE 方式

BPE[13),14)] は，以下のような性質を持つことが知られている．

- 圧縮率は ZIP，LZH より少し劣る．
- 展開処理が高速で，デマンドページングなどに適している．
- 圧縮処理は遅い．

BPE の基本アルゴリズムは，2 バイトデータを 1 バイトデータで置き換えることでデータを圧縮する方式である．**図 2** にその動作の例を示す．オリジナルデータ中の 2 バイトのペアで最も多いものとして AB を選択し，使っていない 1 バイト（以後，トークンと呼ぶ）すなわち X に置き換えるという処理を行う．次に同様に最も多く現れるペアである XC を使っていないトークン Y に置き換える．このような処理を繰り返すという単純な方式で圧縮する．そのため，圧縮時間がかかるという欠点があるものの，展開はビット演算などを必要とせず高速である[16]．

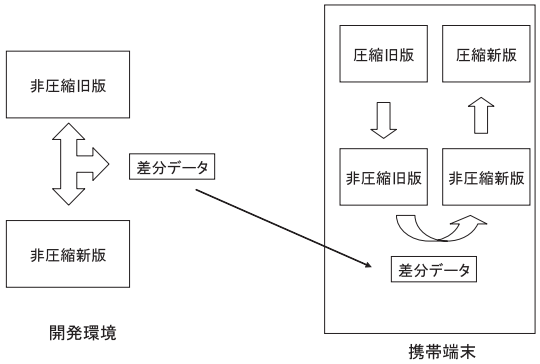

図 3　差分更新
Fig. 3　S/W updating by binary diff.

3.4　圧縮データと差分更新

3.4.1　定性的特性

わずかな不具合を修正し，非圧縮状態では旧版と新版の間に大きな差異がでない場合でも，圧縮状態になると差異が大きくなることが多い．これは BPE に限らず一般に圧縮データに関する特徴である[18]．そのため，圧縮状態の旧版から差分適用して新版を作成しようとすると差分データのサイズが大きくなる．

差分データサイズを小さく抑えるために，**図 3** に示すように差異の小さい非圧縮状態で差分抽出を行い，端末上で圧縮状態にある旧版を RAM 上に展開し，差分適用後の非圧縮状態の新版を圧縮して保存する．更新時間は，データ展開時間とデータ圧縮処理の時間が余分に必要となる．BPE では，展開は高速であるが，圧縮処理が遅いという欠点が大きな問題となる．

3.4.2　実データによる問題点抽出

最近の機能を満載した携帯端末の S/W イメージを利用し，NAND 型フラッシュメモリ上に BPE の形でプログラムコードを置き，ソフトウェア更新機能を動作させる実験を行った．**表 1** に実験に用いたソフトウェアイメージの規模を示す．

端末上でのソフトウェア更新機能は以下の手順で動作する．

(1)　NAND 型フラッシュメモリから圧縮状態の旧版プログラムの RAM への読み込み．
(2)　RAM 上での圧縮状態からの展開による旧版プログラムコードの生成．
(3)　差分データを旧版プログラムコードに適用し，新版プログラムコードの生成．

表 1　ROM イメージサイズ（KByte）
Table 1　The size of binary image (KByte).

	$V1$	$V2$	$V3$	$V4$
全体	59,716	59,716	59,720	59,720
対象外	6,996	6,996	6,996	6,996
対象圧縮状態	52,720	52,720	52,724	52,724
対象非圧縮状態	85,100	85,100	85,100	85,100

表 2　変更ブロック数とページ数
Table 2　The number of updated pages.

更新内容	変更ブロック数	変更ページ数	展開後サイズ☆
$UD1$	361	13	54,008,932
$UD2$	324	129	54,179,307
$UD3$	468	4,812	62,141,778

☆ 展開後サイズの単位は byte

(4)　新版プログラムコードの圧縮．
(5)　圧縮状態の新版プログラムコードで，RAM から NAND 型フラッシュメモリに対して変更のあった更新ブロック[⋆1]の保存．
(6)　保存した圧縮状態の新版プログラムのベリファイ．

実験にはバージョンアップを 3 段階で行ったことを想定し，4 つのバージョンを利用した．それぞれ $V1$，$V2$，$V3$，$V4$ とする．それぞれの ROM イメージのサイズを表 1 に示す．また，$V1 \rightarrow V2$，$V1 \rightarrow V3$，$V1 \rightarrow V4$ の更新をそれぞれ，$UD1$，$UD2$，$UD3$ と呼ぶこととし，**表 2** にそれぞれの更新における変更のあったブロック数[⋆2]とページ数[⋆3]を示す．ブロック数は書き換えたブロック数を示し，ページ数は変更があったページ数である．ここで，ページには変更がなくても位置がずれたりした場合には書き換えが発生する場合がある．

$UD1$，$UD2$，$UD3$ の順に更新内容が大きくなり，修正量も多くなる．フラッシュメモリの更新では同一のページ数の更新でもそのページがどの程度同一のブロックに入るかによって実行時間が変わることがある．そのため表 2 では変更ブロック数と変更ページ数の双方

⋆1 NAND 型フラッシュメモリは複数のページからなるブロック単位でデータを消去する．そのため，わずかな変更であってもブロック単位で消去し，ブロック内の全ページを書き戻す必要がある．
⋆2 1 ブロック 128 ページ
⋆3 1 ページ 4 K バイト

表 3　基本性能
Table 3　Basic performance.

項目	性能
BPE（圧縮）	60.7 KB/秒
BPE（展開）	13.16 MB/秒
NAND フラッシュ書き込み	2.13 MB/秒
NAND フラッシュ読み込み	4.27 MB/秒
差分適用	20.49 MB/秒
ベリファイ計算☆	98.36 MB/秒

☆ ベリファイ計算は NAND 型フラッシュメモリの読み込み部分を除外した性能

表 4　更新時間（秒）
Table 4　Updating time (sec).

	$UD1$	$UD2$	$UD3$
旧版読み込み	14	14	14
旧版展開	0	0	1
差分適用	3	3	3
新版圧縮	1	9	317
新版書き込み	21	19	27
ベリファイ	14	14	14
全体	43	49	376

を示した．変更ページ数と変更ブロック数の関係が逆転する場合などがあるのは，一部のページを修正すると位置がずれてしまい，なんの変更もないのにもかかわらず位置ずれの調整を行うという操作が入るケースがあるためである．このため書き換えるブロック数が多くなる場合がある．

また，最近の機能満載の携帯電話機種のソフトウェイメージデータを適用し，BPE の圧縮速度，展開の速度，NAND 型フラッシュメモリへの書き込みと読み込み，差分適用，ベリファイの各要素の基本性能を測定した．その結果を**表 3** に示す．この基本性能から実データでの更新時間を計算した結果の各段階ごとの実行時間を**表 4** に示す．

結果として，$UD3$ の場合に，極端に圧縮時間が大きくなることが分かった．トータルで 7 分強の携帯端末を利用できない時間があるのに対してその中の 6 分が圧縮の時間であるため，更新時間が長くなるようなケースを改良するには圧縮の高速化が必須であることが確認できた．なお，$UD1$，$UD2$ は修正量が小さく，圧縮時間には影響はあまりない．よって，修正が広い範囲にわたる場合が問題となる．

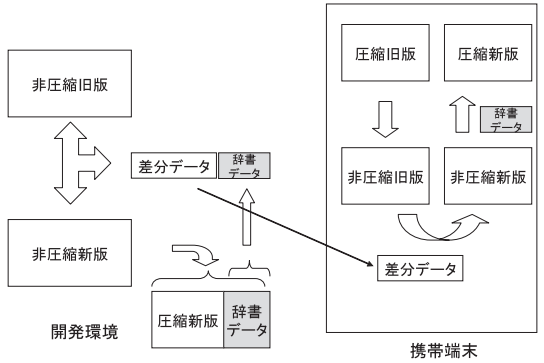

図 4　提案方式
Fig. 4　Proposed method.

また，表 3 における BPE 展開速度は差分適用速度より遅いにもかかわらず，表 4 における BPE 展開時間が早くなり，矛盾しているように見えるが，BPE の適用対象は圧縮してあるため小さく，差分適用対象は非圧縮のため大きい．そのため実行時間では差分適用の方が大きくなっている．

4. 提 案 方 式

4.1　基本的なアルゴリズム

更新時間の大半を占める BPE の圧縮処理は，最大出現頻度のペアを見つける処理とペアを使っていないトークンに置き換える処理の繰返しである．このペアとトークンの組合せを辞書として持ち，展開時にはこの辞書を利用して展開する．

圧縮時にはこのペアを見つける処理で何度もデータを走査するため最も時間がかかる．そこで，**図 4** に示すように，新版と旧版の圧縮前の差分データである更新データに圧縮用の辞書情報を付加しておき，辞書情報を利用して圧縮処理することにより繰り返し走査することを防ぎ高速化することができると考える．

ただし，辞書データを付加するため更新データが大きくなるという欠点を持つ．しかし，基本サービスの停止時間を短くできるとういう観点から更新データ量のある程度の増加は許容できると考える．

4.2　実 現 方 式

クロス開発環境で新版を圧縮後，その圧縮データ内の辞書からトークンとペアの組合せを

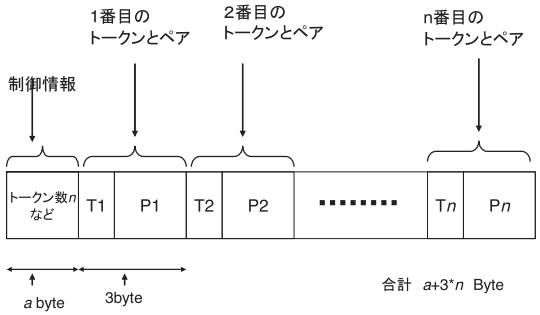

図 5　提案方式の実現
Fig. 5　Implemetation of proposed method.

抽出し，**図 5** に示すように先頭からトークン数などを含む固定長の数バイトの制御情報およびトークンとペアを順番に保持する．

この辞書を利用し，携帯端末上での圧縮処理では，最も多いペアを見つけるという走査を省き，ペアをトークンに置き換えるという処理のみを実行する．ここで，圧縮の際に適用する順序はクロス開発環境上でも携帯端末上でも一致した方がよい．違う順序で適用しても圧縮は可能であるが，圧縮率が劣ってしまうことになるためである．そのために順序を保持する．クロス開発環境上での圧縮と同じ順にデータを置き換えていくことにより元と同じ圧縮状態を作成することができる．

更新データに追加される辞書のサイズは，ページ単位に圧縮しているため，次式で示されるバイト数が更新のあったページ数分加わる．トークン数を n，固定長の数バイトの制御情報を a とする．

$$DictionarySize = a + (n) * 3 \quad (1)$$

よって辞書のサイズはトークン数 n に比例して大きくなる．

4.3　提案方式の測定と評価

表 5 に前節と同じ携帯電話ソフトウェアイメージを利用して評価した圧縮速度を示す．約 6 倍の高速化となることが分かった．従来の方式で問題となった $UD3$ のパターンに対する更新時間を**表 6** に示す．更新時間全体で，376 秒かかっていたのが，112 秒で更新が完了することが分かり，十分効果があることが分かった．

トレードオフとなる更新データの増加を**表 7** に示す．その結果更新データサイズが従来方式と比べどの程度大きくなったかを**表 8** に示す．対象外と記載している部分はどうしても高速化が必要で，圧縮すらしない部分である．辞書データは更新データの約 2 倍弱程度になっている．少々のデータの増加であれば問題がないが，3 倍となるとキャリア側にとってはネットワーク負荷の問題が発生し，ソフトウェアの更新費用をメーカで負担する場合にはコスト的にも問題となることが分かる．ただし，圧縮状態で差分を抽出すると，$UD3$ の場

表 5　提案方式の圧縮性能
Table 5　Performance of proposed compression method.

	従来方式	提案方式
圧縮速度（KB/sec）	60.7	364.3

表 6　提案方式更新時間（秒）
Table 6　Updating time by proposed method (sec).

	$UD3$（従来方式）	$UD3$（提案方式）
旧版読み込み	14	14
旧版展開	1	1
差分適用	3	3
新版圧縮	317	53
新版書き込み	27	27
ベリファイ	14	14
全体	376	112

表 7　提案方式による更新データ増加
Table 7　Increasing updating data size by proposed method.

項目	値
トークン数合計（4,812 ページ分）	458,593
1 ページあたりのトークン数（平均）	95.3
1 ページあたりのトークン数（最大）	205
1 ページあたりのトークン数（最小）	3
辞書サイズ（4,812 ページ分）	1,385,403 byte

表 8　提案方式による更新データサイズ
Table 8　Updating data size by proposed method.

$UD3$	従来方式	提案方式
対象外	220,964	220,964
対象領域	522,245	522,245
変更ページ情報	19,248	19,248
辞書データ	0	1,385,403
更新データサイズ	743,209	2,147,860 byte

合で約 9 M バイトにもなるため，差分更新を使った方がよいことは変わらない．

一方，2 章でも述べたデータの増加をともなわない文献 17) の方式では圧縮速度はメモリの性能などにも依存はするものの，10 倍から 20 倍で実現できることが分かっている．文献 17) の方式はデータの転送量も増加しないため非常に有効であり，圧縮時間は，15 秒から 30 秒程度になり，総合的にみても 1 分強い時間程度で書き換えが終了することになり，提案方式よりも圧縮速度とデータ転送量の面から有効である．

しかし，文献 17) の方式では圧縮率が 1 割程度悪くなる．そのため想定するような組み込み機器の環境ではページの読み込み時間に 1 割近く影響が出ることになり，出荷後に数回利用するであろう機能を高速に行うために，通常利用時への性能影響を出すことは大きな問題であり，このような機器の BPE には単純には適用することができない．

5. 改 善 提 案

ここまでの検証で付加する辞書データ量が多いことが分かった．このデータを削減する方法を検討する．以下の手法がまずは考えられる．

(1)　辞書データを圧縮すること．
(2)　旧版辞書と新版辞書の差分をとって差分データを付加すること．
(3)　圧縮にかかる時間によっては，圧縮用辞書を送らないページを指定するページ選択辞書転送方式．

これらの中で，辞書データの圧縮に関しては，圧縮データは通常辞書そのものも含むため，これ以上圧縮できないことが明白であり，また旧版辞書と新版辞書の差分をとる手法においても，この差分が小さければ元の差分も小さいと想定されるため，効果がでないことが明白である．そのため，圧縮にかかる時間によっては圧縮用辞書を送らないページを指定するページ選択辞書転送方式を中心に検討する．

5.1　ページ選択辞書転送方式

ページごとに辞書データを用いるべきか，そうでないかを分けることによって，更新時間の保障をしたうえで，転送データ量をできる限り少なくする手法を検討した．

ソフトウェア更新の運用の前に，実機を用いて，圧縮時間を含めないページごとのソフトウェア書き換え時間を測定することにより，最低限ソフトウェア書き換えにかかる時間を測定する．その結果から圧縮に要して良い時間を決めることができる．その圧縮時間内に入る範囲でデータの転送量を最小にすることが目標となる．

そこで，次の 4 つの方法に関して実際に比較評価した．

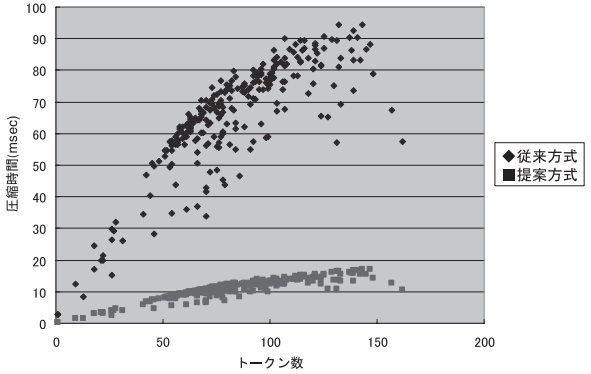

図 6　圧縮時間とトークン数
Fig. 6　Compression time and the number of token.

(1)　まったくランダムにページを選び，順に辞書を転送しないページとし，許容範囲までのページを辞書を転送しないページとして選ぶ方式．ランダムの例として最初から順にページを選択してく．
(2)　転送辞書サイズ順にソーティングし，辞書サイズの小さいものを優先して辞書を転送しないとする方式とする．
(3)　辞書サイズのわりに圧縮に時間がかからないページを優先して辞書を転送しないとする方式とした．
(4)　転送辞書サイズ順にソーティングし，辞書サイズの大きいものを優先して辞書を転送しないとする方式とする[⋆1]．

辞書サイズの割りに圧縮時間のかからないページの選択は以下のように考えた．まず，圧縮時間とトークン数の関係をサンプルデータをもとに調べた結果を**図 6** に示す．縦軸が圧縮率，横軸がトークン数であり，各ページごとに該当する部分を従来方式，提案方式でそれぞれプロットしている．従来方式，提案方式に関係なくトークン数が多いと圧縮時間が増加する傾向であることが分かる．これは，トークン数と変換回数が比例することによるが，多少のばらつきがある．このばらつきに着目するとトークン数が多い，すなわち辞書が大きいわりには圧縮時間はそれほど大きくないページと，その逆のページがある．これらの比率を

⋆1 悪い例と考えて比較のために評価する．

表 9　ページ選択方式ごとの辞書サイズと圧縮時間
Table 9　Dictionary size and compression time for each method of page selection.

圧縮時間（秒）	ランダム方式（Byte）	辞書サイズ順（Byte）	効果順（Byte）	辞書サイズ逆順（Byte）
1	59,079	57,954	57,438	60,429
2	54,309	52,224	51,474	56,928
3	49,581	46,959	46,170	53,109
4	44,211	42,084	40,848	48,972
5	38,865	37,497	36,093	44,979
6	33,414	32,838	31,701	40,551
7	28,650	28,185	26,895	36,324
8	24,300	23,637	22,662	31,710
9	20,607	19,461	18,414	27,084
10	16,821	15,444	14,421	22,524
11	13,410	11,274	10,644	17,700
12	9,747	7,425	7,053	12,486
13	4,854	3,891	3,606	6,825
14	237	180	165	957

利用して選択する．

5.2　各方式の差異評価

評価は携帯電話の S/W イメージ全体のうち，一部の 256 ページ分で評価を行った．評価では，それぞれ 1 秒から 14 秒以内に圧縮時間が終えられるかどうかという観点でデータ転送量がどうなるかを調べた．

各方式の評価結果を**表 9** および**図 7** に示す．縦軸が転送量であり，横軸が圧縮にかかる時間である．比較のために悪い例として，辞書のサイズを逆順に選ぶ場合も示した．辞書サイズのわりに圧縮の時間がかからないページをそのまま転送する方式が最も良いことが分かるが，転送量として全体で考えても，それほど大きな差とはいえず，辞書の大きさで選択するか，ランダムに選んで選んでもよいことが分かる．ただし，ランダムに選ぶ場合でも，各ページで圧縮にかかる時間と辞書のサイズはあらかじめ調べておかないと更新時間を保証できなくなるため，どの方式でも手間は同じである．

結論として，ページの選択方式はどの方式にせよ，一定の目標時間でソフトウェアの更新を終え，データ転送量も比較的抑えることができることを示している．なお，これらの制限時間はソフトウェアの規模やサービスによって決めることになる．各方式による差は若干あり，圧縮時間とトークンの関係を利用する方式が優れているが，処理全体の時間やデータ転送量からみるとわずかな違いでしかないため，どのようにページを選んでも大差ないと考えられるため，実装の楽な方式を採用すればよいと考える．

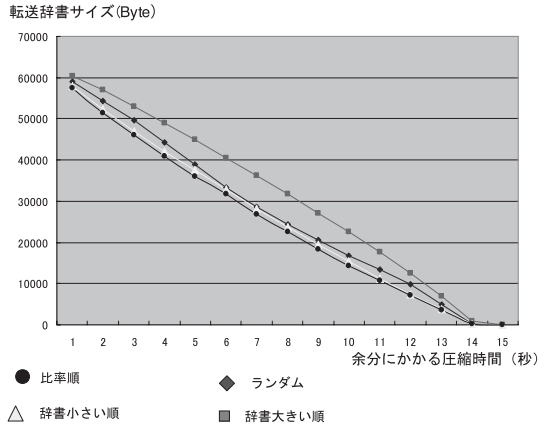

図 7　ページ選択方式ごとの辞書サイズと圧縮時間グラフ
Fig. 7　Dictionary size and compression time for each method of page selection.

6. おわりに

今後普及してくるであろう，NAND 型フラッシュメモリを搭載し，デマンドページング方式を採用する携帯端末において，ソフトウェア更新の問題点として端末上での圧縮速度の面があることを示した．また，この課題に対して，高速化する方式を提案し評価した．提案した方式では約 6 倍の圧縮速度を実現できることが分かった．

しかし，更新用のデータが辞書データの増加により，約 3 倍弱になることが分かり，その改良方式を提案，評価した．改良方式では携帯端末として基本サービスができなくなる時間を一定の時間に抑えたうえでデータ転送量を削減できることを示した．

最近のソフトウェア更新サービスではユーザの気づかないうちに自動的に更新するケースも増えてきている．ダウンロード中にはユーザが利用しようとしても基本機能は問題なく使えるため，ダウンロード時間が長くても影響は少ない．しかし，書き換え中はユーザは基本サービスを利用できない．そのため，ユーザへのサービスという観点からは書き換え時間を短くすることが望ましい．そういう観点では一定時間内にソフトウェアの書き換えを行うことを保障する今回提案の手法は十分効果がある．

一方で，メーカやキャリアの立場ではデータ通信量が大きくなるのは好ましくない．そこで，今後さらに辞書データの特性に注目したうえで，更新データに付加しなければならない辞書データの削減を検討していく予定である．また文献 17) の方式はデマンドページの対象でない部分に対しては有効であると考えるため，複数の方式の組合せなども今後検討していく予定である．また，一般に圧縮率を低下させた場合のデマンドページングへの影響の評価も行いどの程度までが許容範囲かなどの明確化も行う予定である．

参 考 文 献

1) Hoshi, S., Ichinose, A., Nose, Y., et al.: Software update system using wireless communication, *NTT DoCoMo Technical Journal*, Vol.5, No.4, pp.36–43 (2004).
2) Innopath: Understanding Firmware Over The Air-FOTA (online). available from http://www.innopath.com/pdf/fota.pdf (accessed 2008-03-21)
3) 杉山泰一：携帯電話のバグを無線ネット経由で修復，日経コミュニケーション，2003 年 8 月，pp.70–72 (2003).
4) 清原良三，栗原まり子，古宮章裕ほか：携帯電話ソフトウェアの更新方式，情報処理学会論文誌，Vol.46, No.6, pp.1492–1500 (2005).
5) Terazono, K. and Okada, Y.: An Extended Delta Compression Algorithm and the Recovery of Failed Updating in Embedded Systems, *Proc. IEEE Data Compression Conference 2004*, p.571 (2004).
6) 日経ビジネス：ケータイのバグを無線で修正ソフトウェア更新の仕組みは?，日経 BP 社（オンライン）．入手先 http://itpro.nikkeibp.co.jp/free/TIS/keitai/20041201/153313/?P=1&ST=keitai（参照 2008-03-20）
7) 清原良三，三井　聡，栗原まり子ほか：携帯電話ソフトウェア更新のためのバージョン間差分表現方式，電子情報通信学会論文誌 B，Vol.J89-B, No.4, pp.478–487 (2006).
8) 清原良三，三井　聡，木野茂徳：組込みソフトウェア向けバイナリ差分抽出方式，電子情報通信学会論文誌 D，Vol.J90-D, No.6, pp.1375–1382 (2007).
9) Symbian: Demand Paging on Symbian OS (online). available from http://www.symbian.com/symbianos/demandpaging/index.html (accessed 2008-03-21)
10) 竹田正幸，篠原　歩：圧縮されたテキスト上のパターン照合—データ圧縮とパターン照合の新展開，情報処理，Vol.43, No.7, pp.763–769 (2002).
11) Lelewer, A.D. and Hirschberg, S.D.: Data Compression, *ACM Computing Surveys*, Vol.19, No.3, pp.261–296 (1987).
12) Beszédes, Á., Ferenc, R., Gyimóthy, T., et al.: Survey of code-size reduction methods, *ACM Computing Surveys*, Vol.35, No.3, pp.223–267 (2003).
13) Gage, P.: A new algorothm for data compression, *The C users Journal*, Vol.12, No.2, pp.23–38 (1994).
14) 門前　淳，安浦寛人：Byte Pair 符号化を用いた命令 ROM 圧縮，情報処理学会研究報告，No.2001-SLDM-101, pp.1–6 (2001).
15) AlgoTrim: Code Compression Library (online). available from http://www.algotrim.com/index.php?page=code-compression (accessed 2008-03-21)
16) Bellaachia, A. and Rassan, A.I.: Efficiency of Prefix and Non-Prefix codes in String Matching over Compressed Databases on Handheld Devices, *Proc. 2005 ACM symposium on Applied computing*, pp.993–997 (2005).
17) Shibata, Y., Kida, T., Fukamachi, S., et al.: Speeding up Pattern Matching by Text Compression, Lcture Notes in Computer Science, Algorithms and Complexity, Vol.1767, pp.306–315 (2000).
18) 河相英典，深澤　司，小谷　亮ほか：組み込み Linux 上のソフトウェア更新方式 (2)—圧縮データへの差分適用，第 66 回情報処理学会全国大会講演論文集，2003:4D-6 (2003).

(平成 20 年 4 月 11 日受付)
(平成 20 年 11 月 5 日採録)

清原　良三（正会員）

昭和 60 年大阪大学大学院工学研究科応用物理学専攻前期課程修了．同年三菱電機（株）入社．昭和 63 年より（財）新世代コンピュータ技術開発機構に出向．平成 4 年三菱電機（株）に復職．組み込み機器のソフトウェア更新，JavaVM 実装方式，組み込み機器の信頼性，保守性向上に関する研究開発に従事．平成 20 年大阪大学大学院情報科学研究科博士後期課程修了．博士（情報科学）．ACM，IEEE 各会員．

三井　　聡

平成 13 年京都大学大学院情報学研究科修士課程修了．同年三菱電機（株）入社．組み込み機器のソフトウェア更新技術に関する研究開発に従事．

沼尾　正行（正会員）

昭和 62 年東京工業大学大学院博士課程修了．工学博士．同大学工学部情報工学科助手，講師，助教授，同大学院情報理工学研究科計算工学専攻助教授を経て，現在，大阪大学産業科学研究所知能システム科学研究部門教授，（独）日本学術振興会学術システム研究センター主任研究員．人工知能，機械学習の研究に従事．人工知能学会，日本認知科学会，日本ソフトウェア科学会，電子情報通信学会，AAAI，ACM 各会員．

栗原　　聡（正会員）

1992 年慶應義塾大学大学院理工学研究科計算機科学専攻修士課程修了．同年日本電信電話株式会社入社．基礎研究所を経て未来ねっと研究所に所属．1998 年から慶應義塾大学大学院政策・メディア研究科専任講師（有期）．現在，同大学環境情報学部非常勤講師．2004 年から大阪大学産業科学研究所知能システム科学研究部門准教授（同大学大学院情報科学研究科情報数理学専攻准教授兼務）．マルチエージェント，ネットワーク科学等の研究に従事．JST CREST 研究員．著書『社会基盤としての情報通信』（共立出版，共著）．翻訳『スモールワールド』（東京電機大学出版局，共訳）．編集『Emergent Intelligence of Networked Agents』（Springer in Computational Intelligence Series）等．博士（工学）．人工知能学会，日本ソフトウェア科学会，ESHIA 各会員．