システムＬＳＩ設計技術　101－1
(2001. 5. 17)

Byte Pair 符号化を用いた命令ROM圧縮

門前 淳　安浦 寛人

九州大学大学院システム情報科学府 情報工学専攻
〒816-8580 福岡県春日市春日公園 6-1
E-mail: {monzen, yasuura}@c.csce.kyushu-u.ac.jp

あらまし
本稿では，組み込みシステムにおける命令 ROM の面積削減を目的としたオブジェクトコードの圧縮手法を提案する．Byte Pair 符号化はデータ圧縮などに用いられているが，原理が簡単であるので，オブジェクトコードの圧縮にも効果が期待される．特に，命令セットやコンパイラの性質と独立な圧縮が行え，デバッグやバージョンアップによるオブジェクトコードの変更にも柔軟に対応ができる．Byte Pari 符号化を使ったオブジェクトコード圧縮の実験をし，ROM 面積の削減に効果があることを確認した．また，他の手法と定性的に比較を行った．

キーワード：マイコンシステム，コード圧縮，ROM 面積削減，組み込みシステム

Code Compression of Instruction ROM by Byte Pair Encoding

Atsushi MONZEN and Hiroto YASUURA

Department of Computer Science and Communication Engineering
Graduate School of Information Science and Electrical Engineering
Kyushu University
6–1 Kasuga-koen,a Kasuga-shi, Fukuoka 816-8580, Japan

Abstract
In this report, we propose an object-code compression method on instruction ROMs in embedded systems. We apply the Byte Pair encoding to the code compression, which has a very simple coding/decoding scheme. In the proposed method, the compression algorithm is independent of characteristics of compilers and instruction sets, and it is also flexible for program changes by debugging and version-up. We demonstrate the effects of the code compression by the Byte Pair encoding on the reduction of ROM area. We also show the comparisons of this method with other approaches from several viewpoints.

Key Words : Micro computer systems, Code compression, ROM area reduction, Embedded System

1 はじめに

組み込みシステムは，携帯電話や自動車部品の制御など幅広く使われている．プロセッサで実行可能なオブジェクトコードは，チップに内蔵されたROMに書き込まれ製品に組み込まれている．組み込みシステムに要求される処理は，複雑化し多様化している．それに伴い，アプリケーションの規模は増加し，ROM面積を増大させている．チップ面積ではROMが多くを占めるため，ROM面積を削減することは，チップのコストを削減する有効な方法である．

オブジェクトコードはプログラムをコンパイルすることで生成される(図1(1))．そして，オブジェクトコードをそのままROMに格納する．オブジェクトコードのサイズを小さくする方法として以下がある．

- プログラムのアルゴリズムを変更する
- コンパイラの最適化処理を変更する
- 命令セットを変更する
- オブジェクトコードを圧縮しROMに格納する

プログラムのアルゴリズムやコンパイラの最適化処理を変更すると，デバッグ処理が必要になる．命令セットを変更すると，コンパイラやプロセッサの依存する部分を変更する必要がある．時間やコストを抑え，オブジェクトコードのサイズを小さくする方法として，ROM圧縮法がある．ROM圧縮法とは，オブジェクトコードを圧縮してからROMに格納し(図1(3))，プロセッサが命令を実行する際にもとの命令に戻す(図1(2))方法である．今までにROM圧縮を使った手法が提案されている[4]．

プロセッサが命令を実行するためには，命令のアドレスを指定し，命令をフェッチする操作が行われる．ROM圧縮により命令が変換されると，プログラム中の論理的なアドレスとROM上の物理的なアドレスに違いが生じる．プロセッサが圧縮したオブジェクトコードを圧縮していないオブジェクトコードと同じようにアクセスするために2つのデコーダを使う．1つは，圧縮した命令を伸張するための命令用デコーダであり，もう1つは，命令のアドレスを圧縮したオブジェクトコードのアドレスに変換するアドレス用デコーダである．伸張処理は時間を小さくするために，チップにデコーダを付加してハードウェアで行う．ハードウェアで伸張処理を行うと，アプリケーションプログラムに変更を加えないためコストの増加を抑えることができる．ROM圧縮により，ROM面積は減少するが，デコーダの面積が増加する．そのためプロセッサの面積を削減させるには，ROM面積だけでなく，デコーダの面積も考慮して評価をする必要がある．

本稿では，既存の圧縮アルゴリズムであるByte Pair符号化をROM圧縮に適用する実験を示す．実験では，オブジェクトコードのサイズやアプリケーションを変更し，ROM圧縮の面積への影響について調べた．その結果，オブジェクトコードの圧縮率はアプリケーションに依存することが分かった．

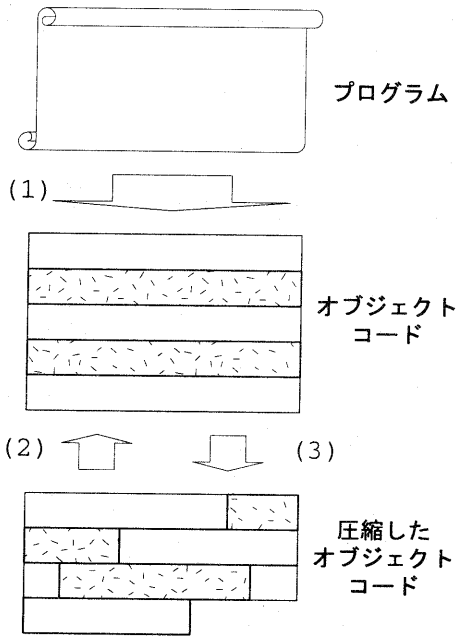

図 1: ROM圧縮の流れ

本稿の構成は以下の通りである．2章でROM圧縮について用語を定義し，ROM圧縮法のモデルから各ROM圧縮手法を分類する．3章では，Byte Pair符号化のアルゴリズムを説明し，Byte Pair符号化を組み込みシステムのROM圧縮に適用する方法を挙げる．4章では，Byte Pair符号化をアプリケーションプログラムのオブジェクトコードに適用する実験を行い考察を述べる．5章では，Byte Pair符号化を使う本手法と，他の手法とを定性的に比較した．6章では，今後の課題を述べる．

2 命令ROM圧縮

2.1 ターゲットシステム

本稿では以下のような特徴を持つプロセッサベースの組み込みシステムをターゲットシステムとする．

- RISCアーキテクチャで，RAM，キャッシュメモリ，ROMをもつ
- アプリケーションは実行可能なオブジェクトコードに変換される
- オブジェクトコードはROMに記録され，製品に組み込まれる

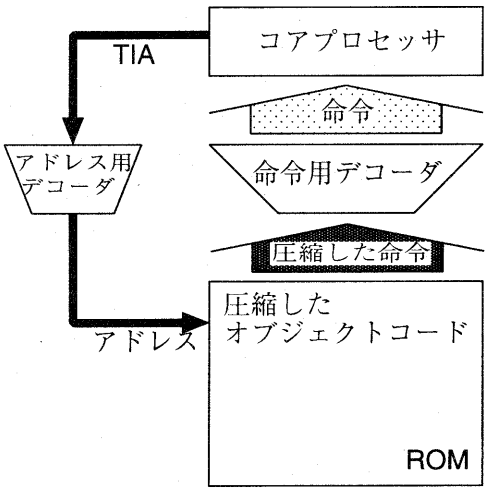

図 2: ROM 圧縮の概要

RISC アーキテクチャなので命令は固定長である場合が多い．ROM 上の命令は製品に組込まれた後は変更されることがない．

2.2 命令 ROM 圧縮法

図 2は ROM 圧縮の概要の図である．コアプロセッサが圧縮された ROM から実行可能な命令にアクセスするために，命令用デコーダとアドレス用デコーダを使う．命令用デコーダは圧縮した命令を伸張し，アドレス用デコーダは分岐先のアドレスを ROM 上の圧縮した命令のアドレスに変換する．命令を圧縮して伸張しても，命令は圧縮前と同じになる必要がある．2 つのデコーダによりコアプロセッサは圧縮の影響を受けないため，コアプロセッサを変更することはない．パフォーマンスの低下を防ぐためには，命令とアドレスのデコード時間を小さくすることが必要になる．

以下の用語を定義する．

- オブジェクトコード：プログラムをプロセッサで実行できる命令に変換したもの
- 圧縮：オブジェクトコードの情報量を失うことなくサイズを小さくする処理
- ターゲット命令アドレス (TIA)：圧縮の前にプロセッサ側から見た命令のアドレス
- 辞書：圧縮された命令列を伸張するための情報
- インデックス：TIA と ROM 上のアドレスを対応させる情報
- デコーダ：圧縮した命令を圧縮以前の命令に変換する回路．辞書を使い圧縮した命令を圧縮前の命令に，インデックスを使いアドレスを TIA に変換する

本稿では，符号化を圧縮の意味で使う．アプリケーションは TIA で ROM へアクセスするため，圧縮によって命令中のアドレスは変更されない．

プロセッサは通常 ROM の連続したアドレスから命令をフェッチし，順次命令を実行する．しかし分岐や例外が起こると，プロセッサは分岐先のアドレスから命令をフェッチして実行する．つまり，プロセッサは分岐先の ROM にランダムアクセスできる必要がある．

圧縮したオブジェクトコード，付加情報 (辞書やインデックス)，デコーダが組み込みシステムに組込まれる．命令の圧縮は組み込みシステムの開発段階で行うため，圧縮処理には十分な時間をかけることができる．一方，伸張処理は組み込みシステムに組込むため，デコーダの面積が小さく，速く伸張できる必要がある．

3 Byte Pair 符号化によるオブジェクトコード圧縮

Byte Pair 符号化はテキストを圧縮する一般に使われているアルゴリズムである [6]．Byte Pair 符号化は圧縮には時間がかかるが，伸張処理が単純であるため速く伸張できる特徴を持つ．本稿では Byte Pair 符号化をオブジェクトコードの圧縮に適用する手法を提案する．

3.1 Byte Pair 符号化

Byte Pair 符号化でデータを圧縮して伸張しても，データは圧縮前と同じになる圧縮方法である．Byte Pair 符号化の圧縮では，テキストに現れていない文字 (未使用の文字) を利用する．Byte Pair 符号化の圧縮は，未使用の文字がなくなるまで，あるいは出現頻度の高い文字の対 (2 つの文字から成る文字列) がなくなるまで，『最も出現頻度が高い文字の対をテキストから探し，その文字の対を未使用の文字に置き換える.』操作を続ける．未使用の文字と文字の対が対応する情報を辞書に登録する．繰り返し文字の置き換えをすることで，置き換えられた文字は 2,3,4 … 個分の文字の情報を持つ．伸張は，圧縮時に置き換えた 1 文字を対応する 2 文字に置き換えることで行う．

図 3の abcabdcab というテキストを例に Byte Pair 符号化の動作を説明する．いまテキストの中では ab という文字の対が 1 番よく出現している．この ab という対を A という文字に置き換え，辞書に登録する．同じようにして，次は cA を B に置き換える．結果として，ABdB という圧縮テキストと A,b の辞書が

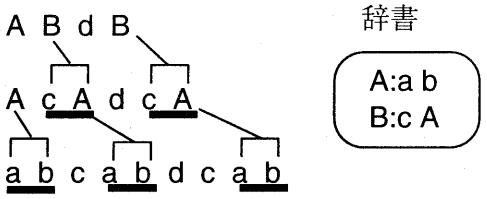

図 3: Byte Pair 符号化の圧縮例

できる．伸張は圧縮と逆の操作をすればよく，文字 A,B が文字 a,b,c,d になるまで置き換えをする．

Given: オブジェクトコード O, where
O中に最も多く出現する 2 文字の文字列 b_i,
$[b_i]$ はその出現回数,
O中に現れない文字 u_j,
jはその個数．

Algorithm Byte Pair 符号化
  while ($[b_i] > 1$)
    if ($j > 0$)
      O中の b_iを u_jに置き換える
      辞書に (u_j, b_i) を登録
      $j = j - 1$
      u_jを更新
      b_iを更新
    else
      exit while
    end if
  end while
  Output 圧縮したオブジェクトコード，辞書
end Algorithm

図 4: Byte Pair 符号化のアルゴリズム

3.2 オブジェクトコードに対する利点

Byte Pair 符号化をオブジェクトコードに適用する利点として以下がある．Byte Pair 符号化はアルゴリズムがシンプルで，圧縮したオブジェクトコードと辞書が分かれている特徴がある．そのため命令 ROM に Byte Pair 符号化を適用する際，ターゲットシステムの性質に合わせ圧縮方法を変更することができる．特徴を以下に挙げる．

- 文字のサイズを変更できる
- 置き換えた文字のサイズと置き換える文字のサイズが等しい
- 特定の命令パターンに依存せず圧縮できる

文字と置き換えた文字のサイズが等しいため，辞書をアラインメントに合わせて ROM 上に保存することができる．

Byte Pair 符号化で圧縮した命令の伸張は，辞書が ROM 上にある場合，インデックスから圧縮命令のアドレスを調べ，辞書へアクセスすることで行われる．そのため，オブジェクトコードのサイズが変わってもデコーダの面積は変わらず，インデックスの面積が変わるだけになる．

Byte Pair 符号化は TIA と ROM のアドレスが違うため，インデックスが必要になる．

Byte Pair 符号化は 2 つの文字を 1 つの文字に割り当てるため，2 つの文字を 1 回の ROM アクセスで読み出すことができれば，伸張処理が速く行える．

4 実験と考察

SuperH[5] の命令に対して Byte Pair 符号化を適用する．Byte Pair 符号化の ROM 圧縮率に対する効果を調べる．

4.1 実験の概要

Byte Pair 符号化を適用するシステムとして，組み込みシステムでよく使われている SuperH プロセッサを選んだ．

実験の対象として，SuperH 用の Linux 上で動作するオブジェクトコードに Byte Pair 符号化を適用した．これらのオブジェクトコードは，SuperH 用のコンパイラの情報，命令セットの情報を含むため，組み込みシステムに使われるオブジェクトコードの代わりに実験に使った．

SuperH のアーキテクチャは 2.1節で述べた組み込みシステムの特徴を持つ．

- RISC アーキテクチャで，RAM，キャッシュメモリ，ROM をもつ
- アプリケーションは実行可能なオブジェクトコードに変換される
- オブジェクトコードは ROM に記録され，製品に組み込まれる

更に，SuperH には以下の特徴がある．

- 命令は 16 ビット固定長
- 32 ビット (2 命令) で命令をフェッチする
- パイプライン化している
- PC 相対アドレッシングモードがある
- 命令中にイミディエイト値がある

命令が 16 ビット固定長だから，命令効率のいいアーキテクチャになっている．プロセッサは，1 回の命令フェッチで 2 つの命令をフェッチできるため，ROM アクセスの回数を少なくできる．そのため，パイプライン化をしたプロセッサではメモリアクセスをしてもストールする回数が少ない．

以下の観点から Byte Pair 符号化の ROM 圧縮率を評価する実験を行なう．

表 1: バイナリのサイズと ROM 圧縮率の関係

バイナリのサイズ (kB)	2293	1024	717	512
ROM 圧縮率 (%)	55.8	59.1	59.4	58.2

表 2: アプリケーションと ROM 圧縮率の関係

アプリケーション	debugperl	perl	dpkg
ROM 圧縮率 (%)	55.8	69.9	56.3

- オブジェクトコードのサイズと圧縮率の関係
- アプリケーションプログラムと圧縮率の関係

本実験での ROM 圧縮率の評価は，圧縮した ROM と，辞書，インデックスの面積で行う．Byte Pair 符号化の伸張は，圧縮命令が符号と判定した場合に辞書をアクセスすることで行なう．命令デコーダの面積は大きくならないから，本実験ではデコーダのサイズは面積へ影響しないとする．Byte Pair 符号化の文字サイズとして，1バイトで行った．SuperH の命令セットは 2 バイトから成るため，1 つの命令が 2 つの符号に割り当てられることもある．

命令 ROM を圧縮した効果を，ROM 面積の圧縮率で評価をする．ROM 圧縮率を，

$$ROM\text{圧縮率}\,(\%) = \frac{\text{圧縮後の } ROM \text{面積} + \text{付加した回路面積}}{\text{圧縮前の } ROM \text{面積}} \times 100 \quad (1)$$

で定義する．

4.2 実験結果

ROM 圧縮率は，バイナリのサイズには影響されていない．ROM 圧縮率はアプリケーションが違うと，ばらつきが見られる．

4.3 実験の考察

ROM 圧縮率は，アプリケーションプログラムに依存し，プログラムサイズには影響されにくいことが分かる．

ROM 圧縮法を組み込みシステムに適用するには，ROM 面積を削減するとともに，パフォーマンスの低下を防止する必要がある．

5 他の手法との比較

本章では他の ROM 圧縮手法と Byte Pair 符号化を使った ROM 圧縮手法を定性的に比較する．3 つの手法を紹介した後に，各手法と Byte Pair 符号化を使った本手法とを比較する．

手法 i：オブジェクトコード中によく出現する命令列を 1 つの命令に置き換える手法 [1]．コンパイラで使われるテンプレートは同じ命令列を多く生成するため，効率よく圧縮が行える．

手法 ii：オブジェクトコード中で使用する命令だけを，プロセッサが提供する命令語長よりも短い命令セットで置き換える手法 [2]．圧縮によりアドレスが不規則にならないため，アドレス用デコーダが必要ない．また，命令のデコード処理を辞書のあるテーブルの参照で行うため高速である．

手法 iii：圧縮アルゴリズムにハフマン符号化を用いた Code Pack という手法 [3]．圧縮した命令のそれぞれにタグをつけることで，圧縮した命令のデコード時間を小さくしている．またデコードの高速化のために，命令をバーストモードで読み出し連続して命令をデーコードしたり，バッファにデコードした命令を保持しデコードした命令を再利用できるようにしたりしている．

表 3: Byte Pair 符号化と他の手法との比較

	A	B	C	D
手法 i	同	同	同	劣
手法 ii	同	優	優	劣
手法 iii	優	同	同	劣

A：圧縮率
B：命令のデコード時間
C：ランダムアクセスの速度
D：プログラムの部分変更への強さ

A:圧縮率については，圧縮のアルゴリズムを元に評価をした．手法 i は，文字列のサイズを 2 にし，繰り返し文字列の置き換えをすると本手法と同じになる．そのため，圧縮率は同じくらいと評価した．手法 ii はアルゴリズムに，ハフマン符号化を使っているため，Byte Pair 符号化よりも圧縮率がよいと評価した．手法 iii は，命令の出現頻度の情報を使ってないため，手法 ii より圧縮率が低いと評価した．

B:命令のデコード時間については，手法 ii がデコーダがシンプルなため優れていると評価した．

C:ランダムアクセスの速度については，手法 ii がアドレスのデコード処理が必要ないために優れていると評価した．

D:プログラムの部分変更への強さについては，各手法ともに部分的な変更によりデータ全体を修正する必要があるために劣っているとした．Byte Pair 符号化を用いる本手法では，変更した部分に変更する以前の辞書を適用できるため，データ全体を修正する必要はない．

6 おわりに

本稿では，Byte Pair 符号化を ROM コード圧縮に適応し，ROM 圧縮率への影響を調べた．

本手法は SuperH 以外のアーキテクチャに対しても適用することができる．また，本手法を適用した際の消費電力やパフォーマンスへの影響を調べることが課題になる．キャッシュやパイプラインがあるモデルへ本手法を適用し，さらなる精度のよい実験をする必要がある．また，複数の実装方法が考えられるが，実装方法による違いを比較することも必要となる．本稿では面積で Byte Pair 符号化の評価を行ったが，更にモデル化の精度をあげることが必要になる．

謝辞

本研究を行うにあたり有益なご助言を頂いた九州大学 大学院システム情報科学研究院 情報理学部門 篠原 歩助教授に深く感謝致します．なお，本研究は株式会社日立製作所の協力で行われたものである．

参考文献

[1] C.Lefurgy, P.Bird, I.Chen and T.Mudge "Improving Code Density Using Compressio n Techniques" Proc. of the 30th Annual International Symposium on Microarchitecture, 1997

[2] 吉田 幸弘, 宋 宝玉, 奥畑 宏之, 尾上 孝雄 and 白川 功 "組み込み用プロセッサの低消費電力化に関する一手法" Proc. 電子情報通信学会論文誌 pages 765-771 1997.5

[3] IBM Corporation "CodePack: PowerPCプロセッサのコード圧縮"
http://www.ibm.co.jp/chips/v5_1/mn51007.html

[4] Charles Lefuragy
"Efficient Execution of Compressed Programs"
http://www.eecs.umich.edu/ tnm/compress/compress-pubs.html
The University of Michigan, 2000

[5] HITACHI
"日立 SuperH RICS engine SH-3,SH-3E,SH3-DSP プログラミングマニュアル"
http://www.hitachi.co.jp/Sicd/Japanese/Products/micom/shmicom.htm

[6] P.Gage
"A new algorithm for data compression"
The C Users Journal,12(2),1994