計算機アーキテクチャ 143－6
（２００１．５．１１）

辞書式コード圧縮支援機構の遺伝的アルゴリズムによる最適化

中西 恒夫 [1)]　　中野 猛 [2)]　　福田 晃 [3)]

[1)] 奈良先端科学技術大学院大学情報科学研究科
tun@is.aist-nara.ac.jp
[2)] リクルート
[3)] 九州大学大学院システム情報科学研究院

概要

本稿では，コード生成時に長い頻出命令列から変換された短い複合命令を，実行時に元の頻出命令列に復元する辞書式コード圧縮支援機構の設計最適化について述べる．また頻出命令列検出アルゴリズムを提案する．頻出命令列検出アルゴリズムで検出された頻出命令列を，コードサイズが最小になるように遺伝的アルゴリズムで選択し，有限のハードウェア資源であるコード圧縮機構に登録する．結果，筆者らが従来使用していた発見的アルゴリズムを用いるよりも，コード圧縮率がわずかではあるが向上した．

Genetic Algorithm Based Design Optimization of the Code Compression Dictionary

Tsuneo Nakanishi[1)]　Takeshi Nakano[2)]　Akira Fukuda[3)]

[1)] Graduate School of Information Science
Nara Institute of Science and Technology
[2)] Recruit Co., Ltd.
[3)] Graduate School of Information Science and Electrical Engineering
Kyushu University

Abstract

Transforming frequent appeared code sequences to shorter composite instructions with hardware assistance such as a code compression dictionary can reduce the code size of the program. This paper discusses design optimization of the code compression dictionary, which expands composite instructions to their original instruction sequences. Moreover, this paper presents a frequent appeared instruction sequence detection algorithm. The frequent appeared instruction sequences detected by the algorithm are registered in the code compression dictionary of limited capacity to minimize the code size with a genetic algorithm. The genetic algorithm reduces the code size a little more than a heuristic algorithm, which the authors have employed so far.

1　はじめに

大容量かつ廉価の半導体メモリが市販される今日においても，組込み機器において，外部メモリの使用はコストや信頼性の点で望ましくないことには変わりはない．また，外部メモリへのアクセスの際には外部バスの駆動に電力を要するため，電池容量に限りのある携帯電話のような組込み機器において，オンチップメモリの活用は必須である．しかしながら，シリコン基板上に確保できるオンチップメモリ

の容量にも制約が存在する．このような背景から古くから研究されているコード圧縮は依然重要な技術である．

コード圧縮は，コンパイラのコード最適化による純ソフトウェア的手法と，ハードウェアの支援のもとに実現する手法とがある．前者の例としては，命令列を意味的に等価なより短い命令列に変換する手法（Strength Reduction）や，意味的に等価な複数のコード断片を関数やブロックにまとめ，それらのコードの出現を関数呼出やジャンプに置換する手法（Procedure Abstruction，Cross Jumping）が代表的である [2, 3]．これらの純ソフトウェア的手法は，多くの場合，実行速度を犠牲にしてコードサイズの削減を図る．一方，後者の例としては，命令語書式の工夫やアクセスできるリソースの制限により命令のビット幅を圧縮する水平方向の圧縮手法や，意味的に等価な複数のコード断片を複合命令に変換する垂直方向の圧縮手法がある [1, 5, 8]．いずれの場合も，圧縮された命令を元の命令あるいは命令列に展開する辞書が，命令フェッチ部やデコーダ部に組み込まれる．

本稿では，垂直方向のコード圧縮を実現する際に必要な，頻出命令列検出アルゴリズム，ならびに遺伝的アルゴリズムによる辞書式コード圧縮機構の設計最適化について述べる．

2　辞書式コード圧縮機構

コードサイズを縮小するコンパイラによるコード最適化技術としては，目的コード中の頻出命令列を関数に変換し，頻出命令列の出現をその関数呼出に置き換える Procedure Abstraction がある．しかしながら Procedure Abstraction は，関数呼出ならびに復帰のオーバーヘッドを生じるほか，キャッシュのヒット率低下の原因となる．

頻出命令列を新たな複合命令として定義し，プロセッサ内部に設けられたコード圧縮支援機構によって，命令フェッチ時に複合命令を元の頻出命令列に展開することにより，Procedure Abstraction のこれらの問題点を回避できる．このようなコード圧縮支援機構は，一般的に命令フェッチ部に辞書として実装され，命令フェッチ部が検出した複合命令をキーに辞書を検索し，元の頻出命令列を引き出してデコーダに送り込む．本稿では，図 1 に示す構造の，辞書式コード圧縮機構の設計最適化を議論する．

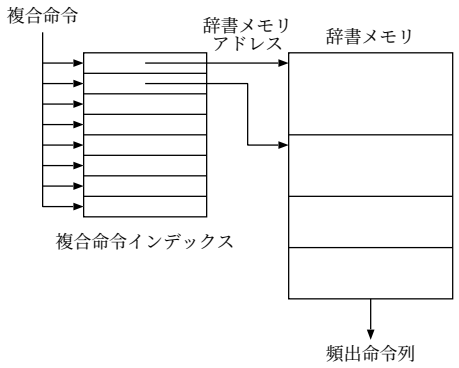

図 1: 辞書式コード圧縮機構

3　頻出命令列検出アルゴリズム

本節では，目的コードから頻出命令列を検出するアルゴリズムを述べる．

提案するアルゴリズムは Fraser らによるコード圧縮アルゴリズムに類似のものである．Fraser らによるコード圧縮アルゴリズム [3]，ならびに提案アルゴリズムは，いずれも頻出命令列検出のために Suffix 木と呼ばれる木構造を生成する．尾部の等しい頻出命令列は，頻出命令列を関数に変換する際に，共通部分に合流するような入口の異なる関数に変換することで，より一層のコードサイズの削減が図れる．Fraser らによる頻出命令列検出アルゴリズムでは，頻出命令列の尾部が等しいかを検査しなければならないのに対して，提案アルゴリズムは Suffix 木の形状から容易に頻出命令列の尾部共通の検査が可能である．

提案アルゴリズムでは，Suffix 木の根を除く個々の節点にひとつの命令とひとつのカウンタを属性として付与している．提案アルゴリズムは，個々の節点 v_n から根 v_0 へ至るパス $\langle v_n, v_{n-1}, v_{n-2}, \ldots, v_0 \rangle$ の節点に属性として与えら

れた命令の列 $\langle c_n, c_{n-1}, c_{n-2}, \ldots, c_1 \rangle$ の出現回数に，v_n のカウンタの値を設定する．したがってカウンタの値の大きな節点に対応する命令列が頻出命令列である．また，各節点の Suffix 木における深さはその節点に対応する命令列の長さと一致する．提案アルゴリズムはこのような Suffix 木を次の手順で生成する．

アルゴリズム 1 【頻出命令列検出アルゴリズム】
プログラムの命令列 $\langle c_1, c_2, \ldots, c_n \rangle$ を入力として受け取り，入力された命令列に対応する Suffix 木を生成する．カウンタの値の大きな節点に対応する命令列が頻出命令列である．

1. $i := n$
2. $i = 0$ ならばアルゴリズム終了．
3. 入力命令列より命令 c_i を取得する．
4. 根に新たなトークンを配置する．
5. 全てのトークン t_j について
   (a) t_j の位置する節点が属性として命令 c_i を持つ子を有する場合，t_j をその子に移動．
   (b) t_j の位置する節点が属性として命令 c_i を持つ子を有しない場合，c_i を属性として持つ子を生成し，t_j をその子に移動する．生成した子のカウンタを 0 にリセットする．
   (c) t_j の位置する節点のカウンタをインクリメントする．
6. $i := i - 1$
7. 2 へ飛ぶ． □

図 2 に命令列 ABCBCABC が入力された場合の Suffix 木の生成過程を示す．図においてハッチングされた節点はトークンが配置されている節点を意味する．

アルゴリズム 1 は，プログラムの命令列の長さを n とすれば，メモリ消費量は $O(n^2)$ であり少なからぬメモリを消費する．しかしながら，筆者らの調査では頻出命令列の分布には局所性が存在し [7]，ある頻出命令列が出現すれば，その出現から遠く離れていない位置に同じ頻出命令列が出現する傾向がある．そのため，アルゴリズム 1 において，ある程度の深さに達したトークンは全て除去することで，すなわちある程度前方までの命令を検査するのみで，十分な頻出命令列を検出しつつ，メモリ使用量を大幅に抑えられる．

4 遺伝的アルゴリズムによるコード圧縮辞書の最適化

アルゴリズム 1 により多くの頻出命令列が検出されるが，図 1 の辞書式コード圧縮機構を構成する辞書メモリならびに複合命令インデックスは有限の資源であり，辞書式コード圧縮機構に格納できる頻出命令列の数とその総量には限りがある．そのためどの頻出命令列を辞書式コード圧縮機構に格納するかでコードサイズの削減量は変動する．そこで本節では，遺伝的アルゴリズムを用いて，辞書式コード圧縮機構に格納する頻出命令列を選別し，コードサイズの最小化を図る．

本稿で扱う問題を以下にまとめる．

- **目的関数:** 辞書式コード圧縮機構に格納された頻出命令列の出現を複合命令に置換した後の目的コードのサイズを目的関数とする．
- **変数:** 目的関数の値を最小化にするような，辞書式コード圧縮機構に登録する頻出命令列の選択を問題の変数とする．
- **制約条件:** 辞書式コード圧縮機構に格納する頻出命令列は最大 n 個，その総量は最大 c バイトとし，これらは設計者によって与えられる．n は複合命令インデックスの要素数，c は辞書メモリの容量に相当する．

4.1 遺伝的アルゴリズムの実装

遺伝的アルゴリズムは，遺伝子とその適合度の定義，ならびに選択，交叉，および突然変異の方法に

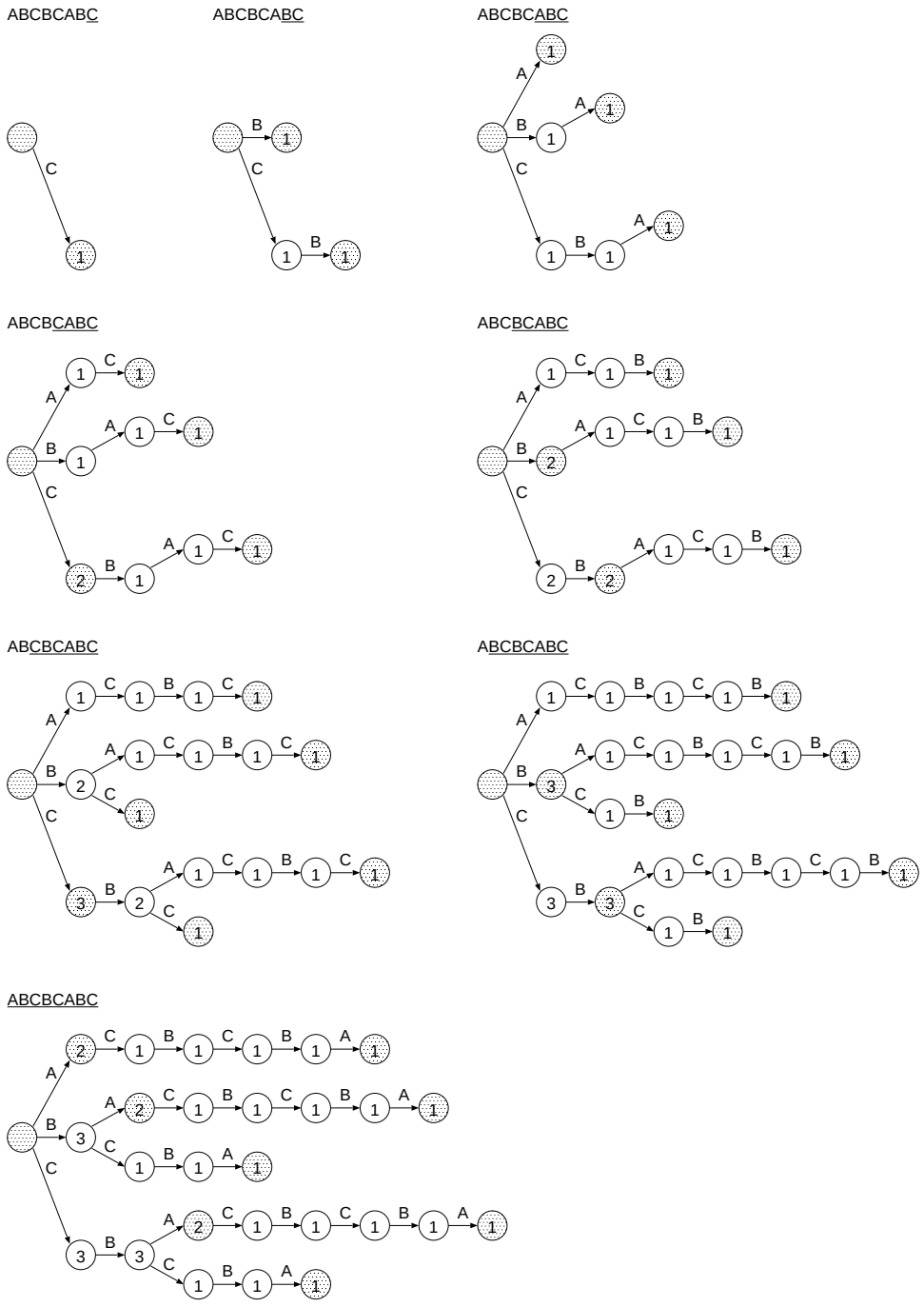

図 2: Suffix 木の生成過程

よって性格づけられる．本稿ではこれらを以下のように定める．

- **遺伝子の定義:** 遺伝子を辞書式コード圧縮機構に格納する c 個の頻出命令列のベクトルと定める．これら c 個の頻出命令列について，遺伝子の頭にあるものから尾にあるものにかけて，順番に複合命令に変換していくものとする．
- **適合度の定義:** 遺伝子の適合度は，当該遺伝子によって規程される通りに頻出命令列を複合命令に置換した後の，プログラムのコードサイズとする．
- **遺伝子の選択:** 遺伝子の選択はトーナメント法 [6] で行う．すなわち，2 本の遺伝子 X，Y を乱数で選び，より適合度の高いほうを次世代に残す．但し，最も適合度の高い遺伝子，いわゆるエリートひとつを必ず次世代に残すものとする．
- **交叉の方法:** 遺伝子は一点交叉法 [6] により交叉する．すなわち，2 本の遺伝子 X，Y を選び，ある確率（交叉率）で，1 以上 c 以下の乱数で指定される位置以降の X の尾と Y の尾を交換する．本稿では交叉率を 0.6 に設定する．
- **突然変異の方法:** 遺伝子は，ある確率（変異率）で，その一部が乱数によって選択された頻出命令列に置換される．本稿では変異率を 0.1 に設定する．

頻出命令列の置換順序によってコードサイズの削減量に変動が生じる．これは頻出命令列の重なりに起因する．例えば，命令列 “ABCDEF” が与えられ，このうち “BCD” ならびに “CDE” が頻出命令列であるとき，先に “BCD” を置換すると “CDE” の部分を置換できなくなり，逆に “CDE” を置換すると “BCD” の部分が置換できなくなる．そのため，上記の遺伝子の定義には，頻出命令列の選択のみならず，置換順序の情報も含めている．

初期遺伝子には，CPack 2000.8 による頻出命令列の選択を指定するものをひとつ含める．CPack 2000.8 は，頻出命令列の長さと出現回数の積が大きいものから順に辞書式コード圧縮機構に格納し，プログラム中の頻出命令列も同じ順番で複合命令に変換する，筆者らが開発した発見的アルゴリズムである．

4.2 適用事例

前小節で実装した遺伝的アルゴリズムを用いて，コードサイズを最小化するようにコード圧縮辞書の構成を最適化する．対象アプリケーションを高速フーリエ変換（FFT）として，n に 8，16，32，64，c に 32，64，128，256 を選択し，意味のある全ての場合についてコード圧縮率を求める．各世代の遺伝子の数は 100 とし，25 世代にわたって交叉を行う．筆者らの実装した CFL（Compiler Framework Library）[4] を用いて，上述の遺伝的アルゴリズムを組み込んだアセンブラを実装し，商用 C コンパイラの出力するアセンブリ言語に対してコード圧縮を行う．筆者らが開発している FPGA ベースの教育用 Z80 互換ソフトコアプロセッサ Freez80[4] に将来的に対応するべく，対象命令セットに Z80 を選択しているが，ここでの議論は他の命令セットにも一般的に適用できる．

表 1 に結果を示す．各欄の上段は CPack 2000.8 を用いて，下段は遺伝的アルゴリズムを用いて，コード圧縮辞書構成を最適化して達成されたコードサイズと圧縮率である．コード圧縮対象としている FFT のプログラムの本来のコードサイズは 2524 バイトである．(但し，ソースが公開されていないため，コンパイラのヘルパルーチンやライブラリは圧縮対象外としている.）遺伝的アルゴリズムを用いて最適化すると，わずかではあるがコード圧縮率が改善される．

5 まとめ

本稿では、メモリ制約の厳しい組込み機器を対象に，コード生成時にプログラム中の頻出命令列を複合命令に置換し，実行時に辞書式コード圧縮支援機構を用いて複合命令を元の頻出命令列に復元することを考え，その際に必要な頻出命令列検出アルゴリズムを提案した．さらに遺伝的アルゴリズムを用いて，この頻出命令列検出アルゴリズムで検出された

表 1: コード圧縮率の改善

	$n=8$	$n=16$	$n=32$	$n=64$
$c=32$	2335 (7.5%)	2335 (7.5%)	NA	NA
	2332 (7.6%)	2326 (7.8%)	NA	NA
$c=64$	2275 (9.9%)	2251 (10.8%)	2251 (10.8%)	NA
	2269 (10.1%)	2233 (11.5%)	2233 (11.5%)	NA
$c=128$	2275 (9.9%)	2146 (15.0%)	2140 (15.2%)	2140 (15.2%)
	2272 (10.0%)	2146 (15.0%)	2140 (15.2%)	2131 (15.6%)
$c=256$	2275 (9.9%)	2146 (15.0%)	2140 (15.2%)	2140 (15.2%)
	2272 (10.0%)	2146 (15.0%)	2140 (15.2%)	2122 (15.9%)

頻出命令列を，プログラムのコードサイズが最小になるように，有限のハードウェア資源であるコード圧縮支援機構に登録する頻出命令列を選択した．Z80 命令セットによる FFT を対象に同遺伝的アルゴリズムを適用した結果，簡単な発見的アルゴリズムよりも，コードサイズを若干削減することを確認した．

しかしながら，本稿で示したデータは，1 命令セットの 1 小アプリケーションのものに過ぎず，遺伝的アルゴリズムの有効性を一般的に証明するものではない．今後は網羅的なパラメータサーベイを行い，当該問題に対する遺伝的アルゴリズムの有効性を明らかにしたい．また，シミュレーテッドアニーリングなど他のメタヒューリスティック手法との比較，整数計画問題としての定式化，CPack 2000.8 よりも強力な発見的アルゴリズムの開発を今後の課題としたい．

謝辞

頻出命令列検出アルゴリズム，コード圧縮アルゴリズムの研究については，中部電力基礎技術研究所・平成 11 年度研究助成（A1 研究）の助成を受けました．CFL のコード圧縮アセンブラフレームワーク部の開発については，情報処理振興事業協会・平成 12 年度未踏ソフトウェア創造事業の助成を受けました．CFL の C コンパイラフレームワーク部の開発については，マツダ財団・2000 年度研究助成の助成を受けています．

参考文献

[1] *An Introduction to Thumb*, ARM Ltd., March 1995.

[2] K. D. Cooper and N. McIntosh, “Enhanced Code Compression for Embedded RISC Processors,” *Proc. of the 1999 ACM SIGPLAN Conf. on Programming Languages Design and Implementation, SIGPLAN Notices,* Vol.34, No.5, pp.139–149, 1999.

[3] C. W. Fraser, E. W. Myers, and A. L. Wendt, “Analyzing and Compressing Assembly Code,” *Proc. of the 1984 ACM SIGPLAN Symp. on Compiler Construction, SIGPLAN Notices,* Vol.19, No.6, pp.117–121, 1984.

[4] *http://t.aist-nara.ac.jp/research/*

[5] M. Game and A. Booker, “CodePack: Code Compression for PowerPC Processors,” *MicroNews,* IBM, Vol.5, No.1, First Quarter 1999.

[6] 伊庭 斉志,「遺伝的アルゴリズムの基礎」, オーム社, 1994.

[7] 中西 恒夫, 中野 猛, 夜久 健一, 福田 晃,「コードサイズ縮小のためのプロセッサと目的コードの協調生成: プロセッサ／目的コードコジェネレータ PinkPanther」, 第 2 回組込みシステム技術に関するサマーワークショップ (SWEST2) 予稿集, pp.67-72, 2000 年 7 月.

[8] 吉田 幸弘, 宋 宝玉, 奥畑 宏之, 尾上 孝雄, 白川 功,「組込み用プロセッサの低消費電力化に関する一手法」, 信学論, Vol.J80-A, No.5, pp.765–771, 1997 年 5 月.