計算機アーキテクチャ 139－24
（2000．8．5）

コードサイズを縮小する組込み向けプロセッサと目的コードの協調生成

中野 猛　　中西 恒夫　　福田 晃

奈良先端科学技術大学院大学情報科学研究科
計算機アーキテクチャ講座（福田研究室）
{takesh-n, tun, fukuda}@is.aist-nara.ac.jp

概要

メモリ要領制限の厳しい組込み機器を対象に，本稿では，入力されたプログラムの目的コードのサイズを最小化するような命令セットを有するカスタムマイクロプロセッサを自動生成し，同時に自動生成されたカスタムプロセッサ上で動作する目的コードを生成する，プロセッサ／目的コードジェネレータ PinkPanther の開発構想について述べる．また，PinkPanther で用いるコード圧縮手法の予備評価を行う．

Cogeneration of an Embedded Microprocessor and Its Object Code to Minimize Memory Consumption

Takeshi Nakano　Tsuneo Nakanishi　Akira Fukuda

Graduate School of Information Science
Nara Institute of Science and Technology

Abstract

In this paper we present a sketch of PinkPanther, a cogenerater which generates application-specific microprocessor and its object code for embedded systems with limited memory capacity. PinkPanther builds an instruction set automatically to reduce memory consumption. We also present preliminary evaluation of the code packing algorithm employed by PinkPanther.

1　はじめに

コストダウンを要求される組込み機器分野において，ソフトウェアを格納するメモリの容量制限は，パソコンなどの一般計算機に対するそれよりも格段に厳しい．さらに近年，システムオンチップの試みが注目を集めているが，ゲート数の問題などによりチップに載せるメモリの容量には限りがある．また，一般的にチップ上のメモリにプログラムを格納し外部メモリ参照のための外部バス駆動を減らすことは，システムの省電力化に有効である．このためソフトウェアのコード自体を圧縮して格納する技術は，半導体メモリが大容量化した今日においても実行時間の短縮と同様，非常に重要なものとなっている．また，組込み機器向けにはさまざまなCPUが用いられるが，それぞれが持つ機能は多くのソフトウェアを動作させるために最大公約数を取った形，つまりどのようなソフトウェアの要求も満足できる機能が実装されている．しかし，特に組込み機器においては，それぞれの機器で実行されるソフトウェアがROMに焼かれて搭載されることが多いため，実際に多くのソフトウェアを実行する必要はない．このような観点から，ユーザがアプリケーションに応じて命令セットを自由に再定義できるようなカスタムマイクロプロセッサが相次いで登場している [1].

前述のようなメモリ容量の限界や省電力化などの観点から，我々は入力されたプログラムの目的コー

ドのサイズを縮小するような命令セットを有するカスタムマイクロプロセッサを自動生成すると同時に，そのカスタムプロセッサ上で実行可能な目的コードを生成するような，プロセッサ／目的コードジェネレータ PinkPanther の開発を進めている．本研究ではその予備実験として，廉価な組込み機器で多く利用されている Z80 プロセッサにおいていくつかのプログラムについて目的コードを生成し，PinkPanther において採用予定の圧縮方式についてどの程度の圧縮効果が得られるかを検証し，成果予測を行う．

本稿では第二節で今まで研究されたコードサイズ圧縮縮小法について検討し，第三節で PinkPanther の開発構想について述べる．第四節で実在する CPU である Z80 に対しての効果予測を行い，最後に第五節でまとめを述べる．

2 従来のコード縮小・圧縮手法

本節では，組込み機器の分野で行われてきたコード縮小・圧縮に関する研究を簡単に紹介し，さらに PinkPanther で用いるコード縮小・圧縮の手法を紹介する．以下では今までの代表的な研究を，コードを実行するハードウェアの変更を許す研究 (ハードウェア的アプローチ) と許さない研究 (ソフトウェア的アプローチ) に分けて考察することにする．

2.1 ハードウェア的アプローチ

ハードウェアの変更を認めるなら，ARM プロセッサ [2] に代表されるように通常の命令セットのサブセットを用意し，アクセスできる資源は制限されるもののコードサイズを縮小する手法が考えられる．

また，テキスト圧縮等の技術を応用する手法も提案されている．これは，CPU コアではそのまま実行不可能な形でコードを ROM に格納し，CPU コアと ROM の間にコード展開機構を介在させる手法である．ここで用いる圧縮手法としては，ミシガン大学の Mudge らが開発している辞書式圧縮手法が注目されている．これは Bird らの研究 [3] から始まり，最近では Lefurgy らの研究 [4, 5] が発表されている．この手法において，圧縮前のコード，圧縮後のコード，辞書の関係は**図 1** に示す形になる．コードサイズだけに着目すると，平均 27%程度の圧縮を行うことができるが，この手法ではプログラムの実行速度が極端に遅くなる問題が指摘されている．

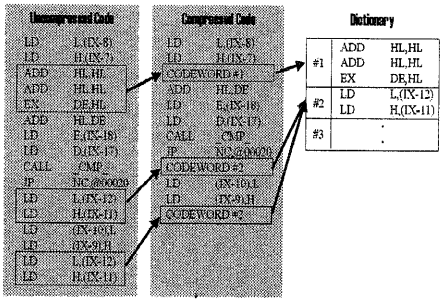

図 1: 辞書式圧縮法概略

IBM 社の PowerPC405 など，辞書式圧縮手法を用いたプロセッサはすでに商品化されている．405 では CodePack[6] と呼ばれる高度な辞書引きを行うことにより，平均 40%程度の高圧縮率を達成している [4].

2.2 ソフトウェア的アプローチ

出力されたオブジェクトコードを解析し，冗長性を排除する研究の多くは，Fraser らの研究 [7] から派生した手法を用いている．Fraser らが提案した手法は Patricia 木 (suffix tree) をつくり頻出命令列を発見し，それらを関数に変換するなどしてコードサイズの縮小を図るものである．VAX を対象とする彼らの研究では，もとのコードよりも 0–39%，平均 7%の圧縮が可能と報告されている．

上記の研究では同じ命令においてもオペランドのレジスタが異なると圧縮することはできない．これに対し Cooper らの研究 [8] ではレジスタの抽象化を行い，さらなる重複部分の発見を行うことで，平均 5%程度の縮小が可能である (ただし，Fraser らの研究とは対象としているプログラムが異なる)．同論文ではさらに実行時間への影響を考慮して，すべての頻出命令列を縮小対象にするのではなく，プロファイルからプログラム中のあまり使われない部分を割り

出し，その部分を対象とする方式も提案されている．

3 PinkPantherの概要

我々は使用メモリ量の削減を目的として，コンパイラの出力から頻出命令列を検出し，それを新たな複合命令として定義し，カスタマイズされたマイクロプロセッサのHDLとそのマイクロプロセッサ用の最終目的プログラムを生成する，プロセッサ／目的コードジェネレータ(開発コード:PinkPanther)を開発している．

PinkPantherはコンパイラならびにプロセッサテンプレートからなるシステムである．本節では最初にPinkPantherコンパイラを用いるプロセッサならびに目的コードの生成過程を追い，続いてPinkPantherプロセッサテンプレートの構造について述べる．

3.1 プロセッサと目的コードの生成

最初にPinkPantherコンパイラは，一般のコンパイラと同様に，入力された原始プログラムを基本命令セットでなる中間プログラムに変換する．PinkPantherプロセッサテンプレート自身は基本命令のすべてを実行可能である．基本命令セットは，数多くの実アプリケーションでの個々のインストラクション使用状況を統計的に吟味した上で設計される．しかしながら，このようにして設計された基本命令セット中のすべての命令を使うことは稀であることが知られている．そこでPinkPantherコンパイラは基本命令セット中から中間プログラム中で利用されていない基本命令を取り除く．さらにPinkPantherはその命令に関するHDL記述をプロセッサテンプレートから取り除く．このことで最終的に得られるカスタムマイクロプロセッサの総ゲート数の削減を図る．

次にコンパイラは中間プログラム中で頻出する命令列を抽出し，新しい複合命令として定義し，最終命令セットを構成する．中間プログラム中の頻出命令列をすべて複合命令に置換することで目的コードを生成する．また同時に新しく追加された複合命令を実行するのに必要な論理回路を合成し，PinkPantherプロセッサテンプレートに追加する．このようにして得られたPinkPantherプロセッサ記述をHDLコンパイラでコンパイルすることにより，PinkPantherコンパイラによって生成された複合命令込みの目的コードを実行可能なカスタムマイクロプロセッサが生成される．

3.2 プロセッサテンプレートの構造

前述の複合命令は，CPU自体が直接実行できる形で論理合成するもの(論理合成型複合命令)と，前節で紹介した辞書式機構を用いて複数の機械語命令に展開するもの(展開注入型複合命令)とに分ける．

パイプラインのIFステージは展開注入型命令を検出すると，それをもとの基本命令列に展開してキューに注入する．展開注入型命令とその元の基本命令列のセットはIFステージに設けられた辞書中に登録しておく．辞書の作成はPinkPantherコンパイラが行う．キューに注入された基本命令列はクロック毎に後続するパイプラインのステージに注入されてゆく．キューが空になるまではPCのインクリメントは停止し，命令のメモリからのフェッチも行わない．

パイプラインのEXステージのみで処理しきれない複合命令は展開注入命令とする．原理上明らかなように，展開注入型複合命令については利用前と後では実行時間に差が出ない．

一方の論理合成型複合命令はPinkPantherコンパイラによって合成された論理回路によってその処理が行われる．パイプラインのEXステージのみで処理しきれる複合命令，たとえば複合論理演算等は論理合成型とする．中間コードの複数クロック分の処理が目的コードでは1クロックで処理されるため，論理合成型複合命令はプログラムの実行時間短縮につながる．

4 予備実験

本節ではPinkPantherを開発するにあたり，まず既存のCPUであるZ80において，展開注入型，また論理合成型複合命令として処理可能な頻出命令列の把握などを目的として予備実験を行う．

予備実験では，yc80コンパイラ(Yellowsoft社製，以下YSコンパイラ)を用いて**表1**に示す6種類のプログラムをZ80用のアセンブラコードにコンパイ

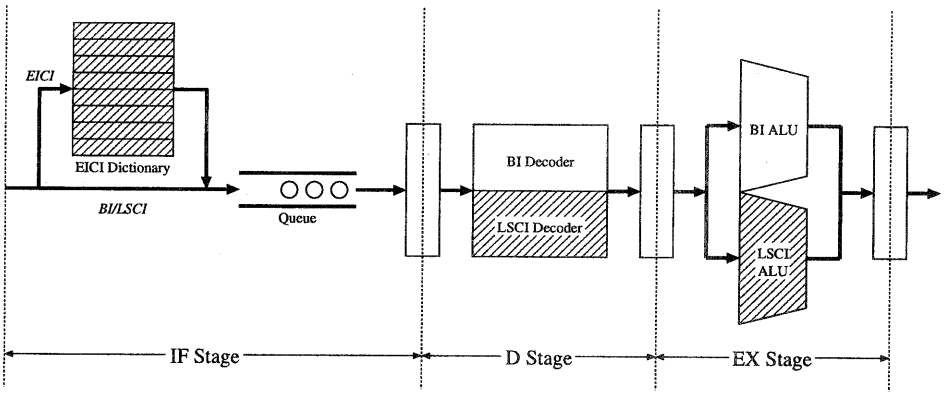

図 2: プロセッサテンプレートの構造

ルし，それぞれのプログラム中で頻出命令列を検出する．この検出には前節で紹介した Patricia 木を用いる．またコンパイラに対する依存性を把握するため，HTC コンパイラ (HI-TECH 社製 demo 版，以下 HTC コンパイラ) を用いて生成したアセンブラコードについても評価を行う．

今回の実験では展開注入型，論理合成型の双方で利用可能な頻出命令列を発見する目的から，2 個から 4 個の連続した頻出命令列のみを検索対象とする．これは辞書内の頻出命令列の粒度を揃える意味もある．

まれに極めて長い頻出命令列が発見されることもあるが，これは関数のインライン展開をしている場合が考えられる．コード圧縮の観点では前出のように，逆に重複部分を関数呼び出しに変換する手法も用いられるが，すでに先行研究があるため，今回は対象外とする．

4.1 個々のプログラムにおいて

まず，すべてのプログラムを yc80 で最適化付きでコンパイルし，個々の中での頻出命令列を検出する．頻出命令列の複合命令置換は，命令列の出現頻度順に行う．ただし，出現頻度が高い頻出命令列についても，実際の置換が 2 回以上行われなければ利用しない．

辞書内に格納できる頻出命令列の数 (エントリ数) を 16,32,64,128 のそれぞれにしたとき，各ベンチマークの圧縮率，すなわちコンパイラに出力結果に対する圧縮処理後のコードサイズを**図 3** に示す．圧縮率は 128 エントリのとき，命令数ベースで 54.8%～64.4% である．このとき，辞書のエントリ数 128 まで調査したが，全てを使い切るベンチマークは des のみであり，他のベンチマークは 64 エントリ程度，特に crc は 32 エントリまでしか利用しないことがわかった．この結果は展開辞書の格納容量コストを考慮しておらず，さらに命令数ベースでの値であるが，PinkPanther において CPU 内に格納する辞書の必要容量として参考になると思われる．

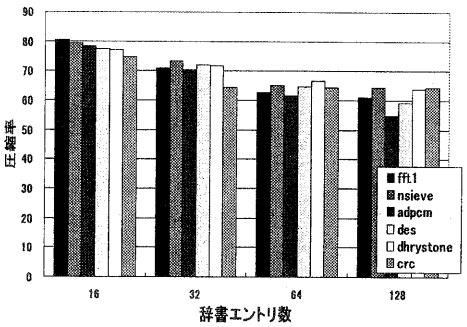

図 3: 辞書サイズの影響

Z80 の標準的なバス幅 (データ 8bit，アドレス

表 1: 今回試したベンチマークプログラム

nsieve	「エラトステネスのふるい」を用いて素数を発生させるプログラム ([9])
dhrystone	一般的なハードウェアベンチマーク ([10])
kmdes	DES 暗号化プログラム ([11])
adpcm	適応差分パルス符号変調アルゴリズム ([12])
fft1	高速フーリエ変換アルゴリズム ([12])
crc	巡回冗長検査アルゴリズム ([12])

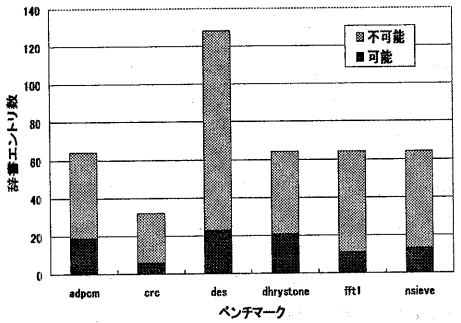

図 4: 論理合成型複合命令による置換可能割合

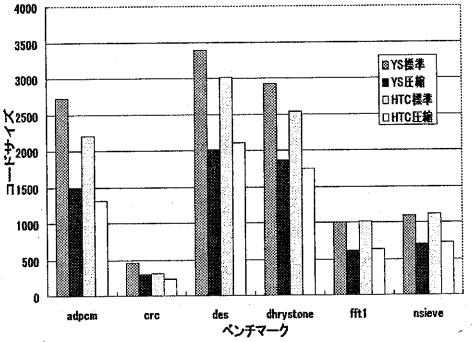

図 5: YS と HTC の圧縮率比較

16bit) は変えず，内部バスの幅は変更可能であるとして，論理合成可能と思われる辞書内の頻出命令列の割合を図 **4** に示す．論理合成できるかどうかの判定は将来的には機械的に行う必要があるが，今回は最大まで利用される辞書サイズの場合について目視で行う．この結果を見ると，今辞書に格納されている頻出命令列のうち，約 22%のものは論理合成可能，つまり論理合成型複合命令として実装可能であることがわかる．論理合成型で実装した場合は，CPU 全体としてのゲート数増加の問題があるが，処理速度などの面ではプラスであり，さらに辞書エントリに空きができた部分に新たな頻出命令列を格納することで，さらなるコードの圧縮が期待される．

次に YS コンパイラと HTC コンパイラそれぞれのアセンブラコードに対する 128 エントリの辞書式圧縮前と後の違いを，図 **5** に示す．

標準で出力されるアセンブラコードの長さとしてみると，比較的 HTC コンパイラのほうが YS コンパイラよりも短いものを出力するが，後者が出力したコードのほうがより大きな圧縮効果が出るため，圧縮後のサイズはほぼ同程度となる．これは HTC コンパイラに標準で搭載されている最適化機能のために，頻出命令列の発見が難しくなっているものと思われる．このことから，コンパイラによって多少辞書式圧縮において効果に差がでることはあるが，辞書式圧縮法を用いることで効果的にコードサイズを圧縮することができることがわかる．

4.2 全プログラムを通して

それぞれのプログラム内で頻出していた命令列の中で，どの程度のものが共有されているかを検証し，結果を図 **6** に示す．

この図から，確かに共有されていた頻出命令列は存在するものの，最も多い crc ベンチマークにおいて

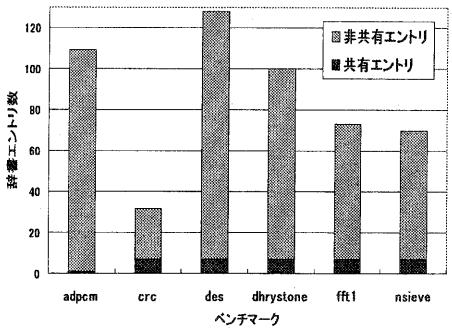

図 6: ベンチマーク間の共有命令列数

21.8%. 平均では約 9.2%と，かなり少ないことがわかる．これは，それぞれのソフトウェアに対するカスタムプロセッサを作るという PinkPanther の方針が有効であることが示された結果である．また，共有されている命令列は，(少ないサンプル数であるが) Z80 命令セットにおいてはどのような命令が追加されるべきか，を知る資料となることが期待される．

5 まとめ

今回の予備実験から，Z80 の命令セットにおいて展開注入型複合命令として実装予定の辞書式圧縮法と，論理合成型複合命令をあわせて用いることにより，従来の研究で行われてきた単一の辞書式圧縮方式よりも優れた圧縮効果を得ることが期待できる．

今後はさらに Z80 以外の CPU が持つ命令セットにおいても調査を行い，見つかった頻出命令列を展開注入型，論理合成型のどちらにおいて処理を行うかの切り分けを機械化する手法を確立する必要があると考えている．

謝辞

本研究は中部電力基礎技術研究所・平成 11 年度研究助成（A1 研究）の助成を受けています．

参考文献

[1] Tensilica. Xtensa Application Specific Microprocessor Solutions. 2000.

[2] Steve Furber. ARM System Architecture. *Addison Wesley Longman*, 1996.

[3] Peter L.Bird and Trevor N.Mudge. An Instruction Stream Compression Technique. *ACM*, 1996.

[4] Charles Lefurgy, Peter Bird, I-Cheng Chen, and Trevor Mudge. Improving Code Density Using Compression Techniques. *IEEE*, 1997.

[5] Charles Lefurgy, Eva Piccininni, and Trevor Mudge. Reducing Code Size with Run-time Decompression. *AAA*, 2000.

[6] IBM. CodePack PowerPC Code Compression Utility User's Manual. *IBM*, 1998.

[7] Christopher W.Fraser, Eugene W.Myers, and Alan L.Wendt. Analyzing and Compressing Assembly Code. *ACM*, 1984.

[8] Keith D.Cooper and Nthaniel McIntosh. Enhanced Code Compression for Embedded RISC Processors. *PLDI'99: ACM SIGPLAN '99*, 1999.

[9] http://ftp.chg.ru/pub/benchmark/aburto/nsieve/.

[10] http://ftp.chg.ru/pub/benchmark/aburto/dhrystone/.

[11] http://rd.vector.co.jp/soft/dos/util/se002367.html.

[12] http://archi.snu.ac.kr/realtime/benchmark/. *SNU Real-time Benchmarks.*