特別研究報告書

曖昧再利用による MP3 エンコーダの高速化手法

指導教官　　富田 眞治 教授

京都大学工学部情報学科

竹村尚大

平成 15 年 2 月 3 日

曖昧再利用による MP3 エンコーダの高速化手法

竹村尚大

内容梗概

近年、小型計算機の普及により、コミュニケーションの手段として用いられる画像や音声の圧縮/伸張技術の高速化が不可欠となってきている。一方、計算機の高速化技術として区間再利用という手法が提案されている。区間再利用とは、一連の命令列を実行する際に、命令区間における入力および出力を記憶しておき、再度同じ入力を用いて実行する際には、記憶していた出力を再利用することにより、命令区間の実行を省略する高速化手法である。さらに、曖昧再利用、すなわち、再利用区間における入力値に許容範囲を設け、取り得る入力パターン数を削減することにより、再利用率の向上を図り、プログラムの実行を高速化する手法が提案されている。特に、画像や音声データにおいては、局所的に類似した値が出現する可能性が高いと考えられるため、圧縮や伸張処理を区間再利用により高速化できる可能性がある。例えば、画像圧縮を行う JPEG エンコーダに対して区間再利用を適用した研究では、圧縮処理においては、入力の微小な変化が出力に大きな影響を与えることがないこと、および、出力画像のわずかな差が視覚に与える影響は小さいことを利用し、入力値を曖昧にすることによる曖昧再利用技術を用いて、最大 40%の命令数削減を実現できることが報告されている。

本研究では、音声圧縮を行う MP3 エンコーダに対して、どのように曖昧再利用を適用し、また、どの程度の高速化が可能であるかについて検討する。画像処理の場合と同様、MP3 エンコーダにおいても、出力の差が聴覚に与える影響は小さいと考えられるため、曖昧再利用が効果的であると予想される。区間再利用の実現には様々な方法が提案されている。本研究では、最近提案されている、コンパイラによる専用命令の埋め込みを必要とせず、既存ロードモジュールの実行時に関数およびループを検出し、再利用することのできる機構を仮定する。また、直接プログラムのソースコード書き換えることにより、再利用効率の良いロードモジュールを生成することを狙う。具体的には、(A) 参照する主記憶アドレス数、(B) 出現可能性のある入出力パターン数、(C) 出現可能性のある再利用区間の数について、それぞれ留意しながら、(1) 再利用区間を特定するための関数切り出し、(2) 参照する主記憶アドレスを減少させるためのバッファ

リング、(3) 関数呼び出しのオーバーヘッドを削減するためのアンローリング、および、(4) 入力パターン数を減少させるための曖昧化などの方法を用いて、曖昧再利用を適用し処理の高速化を図った。

評価および分析には、bladeenc と呼ばれる MP3 エンコーダを用いた。まず、bladeenc の構造を詳細に分析し、再利用可能性および再利用の適用方法について調査を行った。調査の結果、6 ヶ所の命令区間において、コードの書き換えによる再利用効率の向上が期待できることを発見した。次に、それぞれの命令区間について、曖昧化やバッファリングなどの手法を適用した。最終的には、これら 6 ヶ所の命令区間について、合計２０種類の再利用適用手法を考案した。

性能評価においては、まず、それぞれの手法を単独で適用した場合について、再利用の効果を測定し、次に、各手法を組み合わせた場合について測定を行った。単独手法を用いた評価では、bladeenc に再利用を全く適用しない場合に比べて、最大 25%程度の命令ステップ数を削減できる命令区間が見られた。また、bladeenc では、データの量子化処理において大規模な表を用いた高速化の工夫を行っているものの、曖昧再利用により、表を用いなくても bladeenc と同程度の高速化を達成することができた。一方、再利用手法のなかには、適用することにより、かえってオーバーヘッドが増加し、命令ステップ数の削減効果が帳消しになるケースも見られた。

次に、各手法を組み合わせた性能評価では、単独手法の評価に用いた実行命令ステップ数に加え、再利用機構のオーバーヘッドも考慮した全実行サイクル数を用いた比較を行った。また、再利用機構のパラメータの差異による影響、および入力音声の差異による影響についても評価した。この結果、MP3 エンコーダ本来の計算時間と比較して約 4 倍、bladeenc に再利用を適用せずに実行した場合と比較して 20 ~ 25%の高速化が実現できることが明らかになった。再利用機構のパラメータの差異による評価については、大きな再利用表を用いた場合に若干の性能向上が見られたものの、より小さい再利用表を用いた場合でも大きな性能低下は見られなかった。また、入力音声の差異による測定値の差はほとんど感じられなかった。

A Technique to Speedup MP3 Encodeing with Tolerant Reuse

TAKEMURA, Naohiro

Abstract

Recently, the real-time communication becomes important on small and portable computers, so the requirement for high-speed computation of compressing/decompressing image and audio data grows more and more. On the other hand, region reuse technique has been proposed to speed up general purpose computers. Region reuse is a technique which memorizes the input and the output of region while executing the series of instructions, and reuses the result later.

When the program counter encounters the same region with the same input as memorized before, the correct result is retrieved immediately without actual execution. Moreover, tolerant reuse has been proposed that is more aggressive technique to speed the programs based on region reuse. Tolerant reuse gains the effectiveness of region reuse by reducing the precision of the region input values, because the reduction of the input patterns results in higher hit ratio of reuse buffer.

Especially, the image and audio data are considered that the similar values are produced repeatedly within local region, so it will be possible to apply region reuse technique against such region and will be able to speed the compressing and decompressing process.

For example, a research that applies region reuse to the JPEG encoder reported that maximum 40% of instructions were reduced by tolerant reuse technique. The research exploits the fact that the slight differences of input values have little affect on the output values, and the feature that the slight differences of output values have small affect on human.

In this paper, I expected the tolerant reuse on MP3 encoder is effective as in the case of image compressing process, because differences of the output values have little affect on the ears. I describe how to apply tolerant reuse against MP3 audio encoder, and evaluate the effectiveness quantitatively.

Though various implementations of region reuse are proposed, in this paper, I assume the recently proposed hardware which can extract the function/loop from traditional loadmodules and reuse the region automatically without any special instructions controlled by compiler. And by rewriting the source program of MP3 encoder,

I try to generate loadmodules which can be effectively reused. To gain the maximum effectiveness, I pay attention to (A): the number of memory addresses accessed in the region; (B): the number of possible input/output patterns; (C): the number of possible reuse regions. Also, I try to speed up the MP3 encoder by applying tolerant reuse exploiting (1): the function extraction to detect effective reuse region; (2): the buffering technique in order to reduce the number of referenced memory addresses; (3): the unrolling technique to reduce the overhead of the function call; (4): the reduction of the precision of the input vales in order to reduce the number of the input patterns.

I used an MP3 encoder so called 'bladeenc' to analyze and evaluate MP3 encoder. First, I analyzed MP3 encoder deeply, and investigated the probability of region reuse and how to apply region/tolerant reuse on MP3 encoder. As the result, I found 6 regions in which I can expect that the effectiveness of region reuse will be enhanced by improving the source program. Next, against the extracted region, I applied down precision, buffering and so on. Finally I designed 20 methods to apply region/tolerant reuse against the found 6 region.

In performance evaluation, firstly, I measured the effectiveness of region/tolerant reuse in the case of using each single method. Secondary, I measured it in the case of using multiple methods. In the evaluation of the case using single method, compared to the case that no region reuse are applied the bladeenc, 25% instruction steps in the maximum could be reduced. And also, by applying tolerant reuse in spite of using the table with which bladeenc speeds up quantize process, I achieved speed up it at the same level as bladeenc. But, on the other hand, there were some cases that the number of reduced instruction steps were offset by overheads to apply methods.

In the evaluation of the case using multiple methods, we evaluated machine cycles including overheads of reuse machine. Also we evaluated the effect of changing parameters of reuse machine and the effect of changing input audio. As the result, we found that about 75% of machine cycles can be reduced compared to the case of using the huge table in quantize process, 20 to 25% of machine cycles can be reduced compared to bladeenc without reuse. On the other hand, there were no acceptable difference between the cases of using different parameters of reuse machine, and also not between the case of using different input audio data.

曖昧再利用による MP3 エンコーダの高速化手法

目次

第 1 章　はじめに **1**

第 2 章　曖昧再利用技術 **2**

2.1　区間再利用 2

2.2　留意すべき点 5

2.3　再利用区間の切り出し 6

2.4　バッファリング 7

2.5　アンローリング 7

2.6　曖昧化 8

第 3 章　MP3 エンコーダにおける再利用の適用可能性 **8**

3.1　聴覚心理モデルの計算 9

3.2　サブバンドフィルターバンク 9

3.3　MDCT 10

3.4　量子化ループ 11

第 4 章　曖昧再利用の実現 **12**

4.1　$tan^{-1}y/x$ への適用手法 12

4.2　サブバンドフィルターバンクへの適用手法 13

4.3　MDCT への適用手法 14

4.4　量子化処理への適用手法 15

4.5　符号化ビット数計算への適用手法 15

4.6　量子化誤差計算への適用手法 16

第 5 章　性能評価 **16**

5.1　単独の手法ごとの評価 18

5.2　複数の手法を組み合わせた評価 20

第 6 章　まとめ **22**

謝辞 **24**

参考文献 **25**

第1章　はじめに

近年、計算機の小型化および高速化により、画像や音声などのマルチメディアデータを身近な計算機により取り扱うことが可能となってきた。最近では、PDA や携帯端末等の普及により、画像・音声などを頻繁に用いる新しいコミュニケーション手段が注目されてきている。しかしながら、マルチメディアデータのサイズは本質的に大きく、データの圧縮/伸張がが必要である。また、即時性が要求されるコミュニケーションを実現するためには、画像や音声などを高速に圧縮/伸張する技術が不可欠である。

一方、計算機を高速化する技術として、区間再利用 [1][2][3] が提案されている。区間再利用とは、一連の命令列を実行する際に、命令区間における入力および出力を記憶しておき、再度同一入力を用いて同じ命令区間を実行する際には、記憶しておいた出力の再利用により命令区間の実行を省略する高速化手法である。同様の計算を繰り返し実行するプログラムの場合、命令列の実行そのものを削減でき、大幅な高速化を図ることができる。特に、映像や音声などのデータには、局所的に類似した値が出現する性質があるため、区間再利用技術の適用により、大幅な性能向上が期待できる。

マルチメディアを取り扱うプログラムに対する区間再利用の適用に関しては、画像圧縮を行う JPEG エンコーダを用いた研究が報告されている [4]。通常の区間再利用に加えて、入力の厳密な一致を必要としない曖昧再利用が提案されている。再利用区間における入力値に許容範囲を設け、入力のバリエーションを減らすことにより、同じ入力値が出現する確率を高め、最終的に命令区間の再利用率を向上させることができる。[4] では、コンパイル時に区間再利用専用命令を埋め込むことにより区間再利用を実現している。通常の区間再利用の適用では 8%、曖昧再利用の適用では 25 ~ 40%の命令数削減に成功している。曖昧再利用により画質は劣化するものの、劣化の度合は軽微であることが報告されている。

一方、本研究では [3] で提案されている再利用機構を用いる。この再利用機構は、コンパイラによる専用命令の埋め込みを必要とせず、自動的に再利用区間を割り出すことができる。

本研究では、音声圧縮技術の一つである MP3 エンコーダの構造を分析し、[3] で提案されているコンパイラによる専用命令の埋め込みを必要としない区間再

利用技術を利用した高速化の可能性を探る。MP3 エンコーダとは、音声圧縮技術である MPEG AUDIO LAYER 3 (以下 MP3) を用いた音声圧縮ソフトウェアであり、人間の聴覚特性と DCT(離散コサイン変換)、Huffman 符号化を利用することにより、1/10 程度という高圧縮率と高音質を実現している。MP3 は非可逆圧縮技術であり、圧縮の際に情報の欠損が生じることから、入力の誤差が出力に与える影響は少ないと考えられる。また、音声自身に曖昧な要素が含まれるため、音声信号の差が人間の聴覚に与える影響も小さいと予想される。以上ことから、出力の質に大きな影響をおよぼすことなく MP3 エンコーダに対して曖昧再利用を適用できる考えられる。本研究では、専用命令を必要としない再利用機構を前提とし、ソースコードを書き換えるこにより、曖昧再利用を最大限に活用した高速化を図る。

第 2 章では、用いる曖昧再利用技術の概要および留意すべき点について述べる。第 3 章では、MP3 エンコーダの分析および再利用の適用可能性について述べ、第 4 章では、具体的な高速化手法を説明する。第 5 章では、定量的評価を行い、考察を加える。

第 2 章　曖昧再利用技術

2.1　区間再利用

本節では、曖昧再利用の基礎的技術である区間再利用について概要を述べる。区間再利用を実現するためには、入出力を特定できる区間を識別する必要がある。現在提案されているのは、再利用区間の開始および終了を知せる専用命令をコンパイラが生成し、ロードモジュールに埋め込んでおくことによってハードウェアに再利用区間を識別させる方法である。一方、本研究では、SPARC ABI[5] に従って記述されたロードモジュールの実行時に入出力を特定できる命令区間を自動的に識別し、区間再利用を行う機構 [3] を仮定する。以下では、本再利用機構を区間自動検知型 (Automatic Region Detect:ARD) 再利用機構または単に再利用機構と呼ぶことにする。ARD 再利用機構の利点は、コンパイラによる専用命令の埋め込みを必要としないため、既存のロードモジュールをそのまま利用できることにある。特に、命令区間を関数やループといったプログラムの構造と対応させることにより、再利用機構を直接的に用いるプログラミングが可能となる。

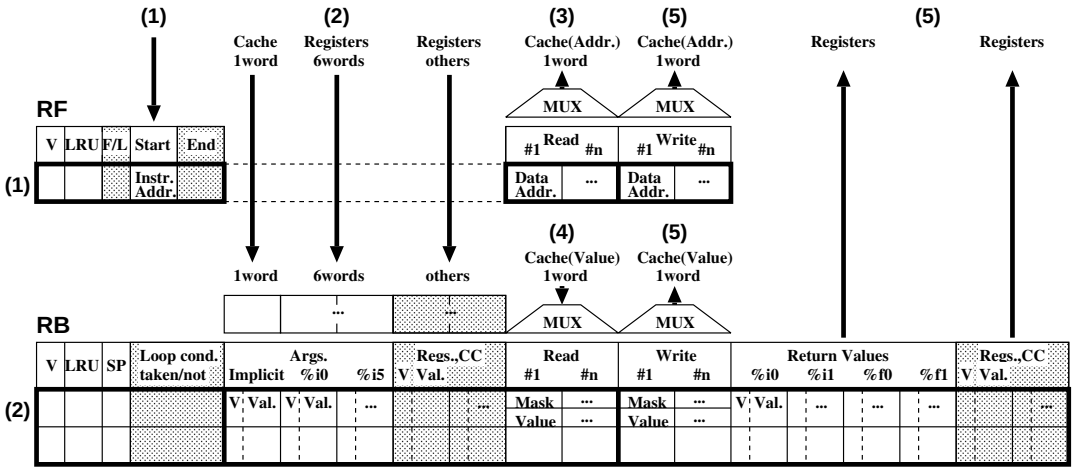


図 1: 再利用表の構成

関数再利用を実現するために、関数管理表（RF）および入出力記録表（RB）を設けることにする。１つの関数を再利用するために必要なハードウェア構成を図 1 に示す（網かけ部分は後述）。複数の関数を再利用可能とするには，本構成を複数組用意する。各表の V および LRU は，各々、有効エントリの表示およびエントリ入れ換えのヒントである。RF は、関数の先頭アドレス（Start）および参照すべき主記憶アドレス（Read/Write）を保持し、RB は、関数呼び出し直前の%sp（SP）、引数（V:有効エントリ,Val.:値）、主記憶値（Mask:Read/Write アドレスの有効バイト,Value:値）、返り値（V:有効エントリ,Val.:値）を保持する。返り値は、%i0 ~ 1（リーフ関数では%o0 ~ 1 に読みかえる）または%f0 ~ 1 に格納され、%f2 ~ 3 を使用する返り値（拡張倍精度浮動小数点数）は対象プログラムには存在しないものと仮定する。Read アドレス（３）は RF が一括管理し、マスクおよび値（４）は RB が管理することにより、Read アドレスの内容と RB の複数エントリを CAM により一度に比較する構成を可能としている。

単一の関数を再利用するには、まず、関数実行時に、局所変数を除外しながら、引数、返り値、大域変数および上位関数の局所変数に関する入出力情報を登録していく。読み出しが先行した引数レジスタは関数の入力として、また、返り値レジスタへの書き込みは関数の出力として登録する。その他のレジスタ参照は登録する必要がない。主記憶参照も同様に、読み出しが先行したアドレスについては入力、書き込みは出力として登録する。関数から復帰するまでに次の関数を呼び出したり、登録すべき入出力が入出力表の容量を越えた、引数の第 7 ワードを検出した、あるいは、途中でシステムコールや割り込みが発生し

たなどの擾乱が発生しなかった場合、復帰命令を実行した時点で、登録中の入出力表エントリを有効にする。以後、関数を呼び出す前に、(1)関数先頭アドレスを検索し、(2)引数が完全に一致するエントリを選択し、(3)関連する主記憶アドレスすなわち少なくとも1つのMaskが有効であるReadアドレスをすべて参照して、(4)一致比較を行う。すべての入力が一致した場合に、(5)登録済の出力(返り値、大域変数、および、Aの局所変数)を書き戻すことにより、関数の実行を省略することができる。

さて、次にループ再利用についてする。関数に含まれる命令は、Call命令の分岐先からReturn命令までである。同様に、ループでは、後方分岐命令の分岐先から、同じ後方分岐命令までである。したがって、この間の入出力を登録しておくことにより、関数と同様にループを再利用することができる。ただし、関数では局所変数の登録を除外することができるものの、ループでは除外することができない。これは、ループ内の局所変数が規定されないためである。すなわち、ループの再利用に必要な入出力は、参照したレジスタおよび主記憶アドレスのすべてである。このため、ループ再利用には、1の網かけ部分を設けてある。具体的には、RFに、関数とループの区別(F/L)およびループの終了アドレス(End)、RBに、ループ終了時の分岐方向(taken/not)、引数や返り値以外のレジスタおよび条件コード(Regs,CC)を追加する。

ループが完了する以前に関数から復帰したり、前節にあげた擾乱が発生するなど、ループの入出力登録が中止されなければ、登録中のループに対応する後方分岐命令を検出した時点で、登録中の入出力表エントリを有効にし、現ループの登録を完了する。さらに、後方分岐命令が成立する場合は、次ループが再利用可能かどうかを判断する。すなわち、後方分岐する前に、(1)ループ先頭アドレスを検索し、(2)レジスタ入力値が完全に一致するエントリを選択し、(3)関連する主記憶アドレスをすべて参照して、(4)一致比較を行う。すべての入力が一致した場合に、(5)登録済の出力(レジスタおよび主記憶出力値)を書き戻すことにより、ループの実行を省略することができる。再利用した場合、RBに登録されている分岐方向に基づいて、さらに次ループに関して同様の処理を繰り返す。一方、次ループが再利用不可能であれば、次ループの実行およびRBへの登録を開始する。

2.2 留意すべき点

再利用機構を明示的に利用してプログラミングを行う場合、(1) 参照する主記憶アドレス数; (2) 予想される入出力パターン数; (3) 予想される再利用区間の数; に留意しなければならい。

第 1 に、RF の各エントリには、命令区間が参照した主記憶アドレスが登録され、一致比較の対象となるため、同一命令区間であっても、実行されるたびに異る主記憶アドレスを参照する場合には、異なる入出力パターンとして、各々異なる RB エントリに記憶される。したがって、参照する主記憶アドレスの数が登録可能な READ/WRITE のエントリ数を越えた場合には、RB エントリに空きがあっても登録することができず、引き続き登録するためには、以前に登録された RB エントリを削除することにより、RF に登録されている READ/WRITE アドレスを減らす必要がある。また、主記憶アドレスが変化しない場合でも、アドレス数そのものが多いと登録することができない。このような状況を防ぐために、再利用を期待する命令区間では、参照する主記憶アドレス数に留意する必要がある。

第 2 に、各 RF に割り当てられた RB エントリをすべて使い切った場合、さらに RB エントリへの登録を行うためには、不要かと思われる RB エントリを追い出す必要がある。すなわち、再利用区間の大きさと再利用率との間にトレードオフが生じる。大きな命令区間が再利用できれば一度に多くの命令を削減できる一方、命令区間を大きくすると入力数が増える傾向にあり、入力パターンが増加することにより、登録すべき RB エントリ数が増加するためである。入力パターン数が RB の最大エントリ数よりも多いと、将来再利用されるはずの RB エントリが追い出され、再利用効率が下がる。したがって、再利用区間の大きさ、特に出現し得る入力パターン数に留意しなければならない。

第 3 に、RF の最大エントリ数を考慮に入れる必要がある。大規模なプログラムを実行する場合、一般に多くの再利用区間が出現する。再利用区間の数が RF の最大エントリ数を越えた場合、古い RF エントリが LRU アルゴリズムによって追い出される。一般に、RF に登録されている区間の中から、再利用効率が最も悪い区間を厳密にハードウェアが特定することは困難であり、再利用区間が多数検出されると、将来有効に再利用されるはずの命令区間が RF から追い出される可能性が高まる。したがって、再利用機構に対し RF からの追い出しを明

示的に禁止する機構が有効であると考えられる。本研究では、関数を再利用区間とする場合に、特定の関数が RF から追い出されないようにする機構を加え、再利用可能性の高い命令区間が常に RF に存在するよう工夫した。

以下では、実際にソースコードに適用するための具体的手法について議論を進める。なお、以下の 4 つの手法においては、いずれも再利用による命令数削減とソースコードの書き換えによるオーバーヘッドがトレードオフの関係にあり、バランスを考慮しつつ、効率の良い高速化手法を模索する必要がある。

2.3 再利用区間の切り出し

ARD 再利用機構が再利用する命令区間は関数およびループであるため、再利用させる区間を関数またはループにまとめる必要がある。この際、留意すべきことは、前述したように、入出力アドレスの数が RF の READ/WRITE のエントリ数を著しく越えないよう調整することである。ループの一部を関数として切り出すことにより、前述の機構を用いて、RF からの追い出しを防ぐことができる。さらに、ループパラメタのような制御変数が存在する場合、基本的に、ループを再利用することはできない。例えば、変数 i がループ A の実行ごとに変化し、ループ内において i をインデックスとする主記憶アドレス $x(i)$ を用いた処理が行われる場合、i は A の入力となるため、A の実行中に同じ入力が出現することはない。しかし、$x(i)$ を用いた処理を、$x(i)$ を入力とする関数 F として切り出すことにより、i が変化しても $x(i)$ が等しければ再利用することが可能となる。

もちろん、関数として切り出す場合は、関数呼び出しのためのオーバーヘッドが余分にかかるものの、再利用による命令数削減の効果がオーバーヘッドを上回れば高速化が可能である。

ループ A の外側にループ B があり、B の繰り返しごとに A が使用する変数に変化がない場合は、A の再利用が可能である。例えば、k 回目の B と l 回目の B において、$x(j)$ の値が同じであれば、$i = j$ における A が再利用できる。ただし、B の命令区間が大きく、B の実行において多数の再利用区間が検出された場合、前述したように、A に関する RF エントリが追い出されてしまい、再利用できない可能性がある。

2.4 バッファリング

ARD 再利用機構では、入出力のアドレスと値の両方が一致しないければ再利用が行われない。例えば、再利用区間 C において、大きさ n の配列 a のうち n/m 個が参照され、C の各繰り返しが参照する n/m 個は互いに異なる主記憶アドレスとする。このような場合、C が参照する n/m 個の値自身がすべて等しくても再利用することができない。この問題を解決するために、C の各繰り返しの直前において、C が参照する n/m 個の値を、一旦大きさ n/m の別の配列 b に移しかえ、C では b の値を参照して計算を行うことを考える。区間 C が参照するアドレスが b の n/m 個のみとなり、一致比較の対象からアドレスを除外することができる。以上の方法をバッファリングと呼ぶことにする。出力先の主記憶アドレスが実行のたびに変化する区間 (D とする) に対してもバッファリングを適用することができる。すなわち、D において出力を固定アドレスに格納しておき、D の外部において正しい主記憶アドレスに移しかえる操作をすればよい。

バッファリングは前述した第 1 および第 2 の留意点に対して有効な手法である。再利用区間が参照する主記憶アドレスの数を減らすことができ、RF の READ/WRITE エントリの溢れを防ぐことができる。また、バッファリング前の区間 C において、a の取り得る値が α 個ある場合、入力パターン数は $\alpha^{n/m} \times m$ となるのに対し、バッファリング後の入力パターン数は $\alpha^{n/m}$ となりることから、RB エントリの溢れを防ぐことができる。

バッファリングでは、主記憶アドレスの値を移しかえるときにオーバーヘッドが生じるものの、再利用による命令数削減の効果がオーバーヘッドの合計を上回れば高速化が可能である。

2.5 アンローリング

入力パターン数が少なく、非常に高い確率で再利用される区間 (E とする) において、ループ制御や関数呼び出しのオーハーヘッドを削減するために、アンローリングを行うことが考えられる。すなわち、E の繰り返しを一つにまとめ、新たな再利用区間とする。r 回の繰り返しを一つにまとめる場合、入力パターン数は r 乗に増加するものの、E におけるオーバーヘッドは約 $1/r$ となる。アンローリング後の入力パターン数が RB に十分収まる程度であり、かつ E の実行回数が多く、アンローリング前と同様に高い確率で再利用されるならば、より

高速化することができる。短い命令区間ほどがオーバーヘッドが占める割合が高いため、アンローリングの効果が高い。

2.6 曖昧化

出力に影響が現れない範囲において入力を曖昧化することにより、入力パターン数を減らし、再利用効率を上げることが考えられる。入力を曖昧化する主な手法には、よりビット数の少ないデータ型へのキャストや、下位 bit の幾つかを無視するものがある。ただし、これらの操作はソースコード上において行うため、オーバーヘッドが生じるまた、ソースコードにおいて、再利用区間の入力曖昧化しようとしても、同じ主記憶アドレスの値が何度も入力として使用される場合は、同じ曖昧化を何度も実行することになり、曖昧化の処理そのものが無駄である。したがって、本来は再利用区間における入力を曖昧化するところを、入力の元にである前段階処理の出力を曖昧化する方が効率がよい。

第3章　MP3エンコーダにおける再利用の適用可能性

本章では、MP3 エンコーダを分析し、どの部分に再利用を適用できるかについて詳述する。本研究で分析、書き換えに用いた MP3 エンコーダは bladeenc ver0.92.0 ((c) Copyright 1998, 1999 - Tord Jansson) である。また、MP3 の技術概要については、[6] を参考にした。

MP3 エンコーダは、音声を周波数 F によりサンプリングした PCM 信号を圧縮し、MP3 形式を出力する。処理内容は、(1) 聴覚心理モデルの計算; (2) サブバンドフィルターバンク; (3)MDCT(窓関数処理+離散コサイン変換); (4) バタフライ演算; (5) 量子化ループ; (6) フレームストリームの作成; からなっている。処理間のデータフローを図 2 に示す。また、bladeenc における各処理時間の割合を表 1 に示す。

各処理は 1 フレームまたは 1 グラニュールという単位の PCM 信号、および、中間結果を入力とする。1 フレームとは 1152 個のサンプリングデータ、1 グラニュールは半分の 576 個のサンプリングデータである。

次に、各処理の内容と再利用可能性について説明する。だだし、(4) バタフライ演算、(6) フレームストリームの作成については、処理時間が短いため、再利用の対象としなかった。

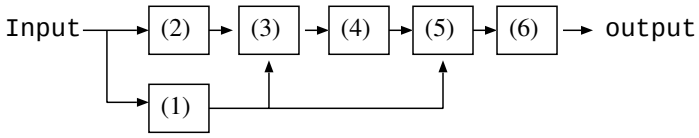


図 2: MP3 エンコーダの処理におけるデータの流れ

表 1: エンコーディング過程における処理時間の割合

処理	割合
聴覚心理モデルの計算	32%
サブバンドフィルターバンク	24%
MDCT+バタフライ演算	12%
Iteration Loop	31%
フレームストリームの作成	1%

3.1 聴覚心理モデルの計算

入力音声 1 グラニュールに対する処理であり、聴覚心理を考慮した計算により、MDCT の窓長や、聴覚影響の受けない量子化誤差と周波数エネルギーの比を計算する。聴覚心理モデル全体における入力データは 576 個の整数であり、n 通りの整数が入力される場合、入力パターン数は n^{576} となる。したがって、全く同じ入力が出現する可能性は低く、聴覚心理モデル全体における再利用の効果は低いと考えられる。さらに内部を調査すると、1024/256 点 FFT および $tan^{-1}y/x$ のそれぞれが聴覚心理モデル全体の約半分の処理時間を占めていることが分かる。1024 点 FFT および 256 点 FFT 全体については、入力パターン数が n^{1024} および n^{256} となることから、再利用の可能性は低い。しかし、FFT の内部のバタフライ演算は 6 個の実数のみを入力とするため、再利用できる可能性はある。また、$tan^{-1}y/x$ についても、実数 2 個のみの入力であるため、再利用可能性は高いと考えられる。

3.2 サブバンドフィルターバンク

入力音声 1 グラニュールに対するの処理であり、デジタルフィルターにより入力信号を 32 個の帯域に分割し、これらを 1/18 にダウンサンプリングする。サブバンドフィルターバンク全体における入力データは 512 個の整数であり、そのままでは再利用可能性は低いと考えられる。そこで、サブバンドフィルター

バンクの内部を検討してみると、式 1 に示す計算が、サブバンドフィルターバンク全体の処理時間の約 80%を占めることがわかる。なお、$M_{i,j}$ は定数、x_j は実数入力値である。式 1 の通りに j についての積和計算全体を再利用区間とした場合、入力は 64 個の実数となり、また、i が 1 から 32 の値であることから、32 組の定数を用いた積和計算となる。したがって、x_j が n 通りの値をとる場合、入力パターン数は $n^{64} \times 32$ となり、再利用できる可能性は低い。しかし、式 1 を、$s_i = s_i + M_{i,j} \times x_j$ と変形し、j を固定し、$i = 1, \cdots, 32$ における $M_{i,j} \times x_j$ の計算を一つの再利用区間と考えた場合、命令区間における入力は x_j のみとなる。$j = 1, \cdots, 64$ であることから、この命令区間で用いる定数は 64 組であり、入力パターン数は $n \times 64$ と減少するため、再利用が可能であると考えられる。

$$s_i = \sum_{j=1}^{64} M_{i,j} \times x_j (i = 1, \cdots, 32) \tag{1}$$

3.3 MDCT

サブバンドフィルターバンクの出力 1 グラニュールに対する処理であり、窓処理および離散コサイン変換による入力信号の無相関化を行う。全体としての入力は 576 個の実数であるものの、1 回の MDCT に対する入力は 36 個の実数であるため、一つの実数が取り得る値が n 通りの場合、入力パターン数は n^{36} となる。さらに MDCT の内部について調査すると、式 2 に示す計算が MDCT の処理時間の 90%以上を占めることがわかる。ただし、W_j, $C_{i,j}$ は定数、X_j は実数入力値である。前節と同様、式 2 の通りに j についての積和計算全体を再利用区間とした場合、入力は 36 個の実数となり、また、i が 1 から 18 の値であることから、18 セットの定数を用いた積和計算になる。したがって、x_j が n 通りの値をとる場合、入力パターン数は $n^{36} \times 18$ となる。一方、前項でも行った式の変形により、$i = 1, \cdots, 18$ における $W_j \times x_j \times C_{i,j}$ の計算を再利用区間とすると、入力は x_j のみとなり、また、$j = 1, \cdots, 36$ であることから、この命令区間で用いられる定数は 36 組となり、入力パターン数は $n \times 36$ と減少するため、再利用の可能が高まると考えられる。

$$s_i = \sum_{j=1}^{36} W_j \times x_j \times C_{i,j} (i = 1, \cdots, 18) \tag{2}$$

3.4 量子化ループ

バタフライ演算の出力 1 フレームに対する処理である。聴覚影響を受けない量子化誤差の範囲内において、Huffman 符号化時に最も効率が良い量子化ステップを計算し、入力信号の量子化を行う。量子化ループ全体の入力はバタフライ演算の出力である実数 1152 個および聴覚心理モデルにより計算された様々なフレームパラメータである。また、内部では Huffman テーブルを参照しており、入力数が非常に多いため、量子化ループ全体の再利用は不可能である。量子化ループ内部の処理において多くの処理時間を占めるのは、量子化処理 (41%) と符号化ビット数の計算 (21%) である。

さて、量子化処理では、式 3 に示す計算が繰り返し実行される。ただし、x は実数、m は量子化ステップ数により決定される値、$nint$ は四捨五入演算である。式 3 の入力は x および m のみであるため、再利用可能性が高い。また、bladeenc では、$nint(X^{0.75} - 0.0946)$ における X および計算結果を保持する 16 万エントリの表 (以下、量子化テーブルと呼ぶ) が用意されており、表の参照による高速化が実現されている。一方、量子化テーブルを参照する代わりに $nint(X^{0.75} - 0.0946)$ を毎回計算する場合では、式 3 の計算に要する時間がエンコード処理全体の約 70%を占める。

次に、符号化ビット数の計算においては、選択された Huffman テーブルを用いて、符号化時の合計ビット数を計算する。具体的には、2 つの量子化されたデータをインデックスとして Huffman テーブルを参照する処理が繰り返し実行される。2 つの数はそれぞれ 0 以上 8206 未満の整数であり、また、実際に出現する値は 100 未満がほとんどを占めるため、再利用可能性は高いと予想される。

その他の処理では、量子化誤差の計算が量子化ループの処理時間の約 10%を占めており、式 4 に示す計算が繰り返し実行される。ただし、x は実数入力、y は x を量子化した値、s は量子化ステップ数である。入力は x,y,s のみであるため、再利用できる可能性が高い。また、bladeenc では、$y^{4/3}$ の計算をすべての y $(y = 0, \cdots, 8206)$ について最初に実行しておくこと、同じ y を用いた $y^{4/3}$ の計算が繰り返し実行されることを避けている。このような手段は本質的に、再利用機構に写像できるため、高速化が期待できる。

$$y = nint((x \times m)^{0.75} - 0.0946) \tag{3}$$

$$t = (x - y^{4/3} s)^2 \tag{4}$$

表 2: 各再利用区間の処理時間と再利用可能性のまとめ

	処理		処理時間	再利用可能性
聴覚心理モデル	1	全体	32(8)	×
	2	FFT のバタフライ演算	16(4)	△
	3	$tan^{-1}y/x$	16(4)	○
サブバンドフィルターバンク	4	全体	24(6)	×
	5	j に関する $M_{i,j}y_j$ のループ	20(5)	×
	6	i に関する $M_{i,j}x_j$ のループ	20(5)	△
MDCT	7	全体	12(3)	×
	8	j に関する $W_jx_jC_{i,j}$ のループ	11(3)	×
	9	i に関する $W_jx_jC_{i,j}$ のループ	11(3)	△
量子化ループ	10	全体	31(82)	×
	11	量子化処理	12(72)	◎
	12	符号化ビット数計算	6(1)	◎
	13	量子化誤差計算	3(1)	◎

以上、MP3 エンコーダの各処理について、再利用可能性を検討した。各処理における処理時間量、再利用可能性を表 2 にまとめた。なお、処理時間はエンコーディング全体の処理時間を 100 とした場合の値である。ただし、量子化処理において、量子化テーブルを使用せずに、$nint(X^{0.75} - 0.0946)$ を直接実行した場合の処理時間を () で囲んでいる。

第 4 章　曖昧再利用の実現

本章では、前章における検討結果に基づき、曖昧再利用を用いた具体的な高速化手法について詳述する。

4.1　$tan^{-1}y/x$ への適用手法

3.1 節であげた、$tan^{-1}y/x$ を計算する命令区間に対して、曖昧再利用を適用する。すなわち、入力 x および y の下位 20bit を無視する。この手法を (A+tol) と呼ぶことにする。

4.2 サブバンドフィルターバンクへの適用手法

3.2 節であげた、式 1 の部分について関数切り出しを行う。bladeenc では、式 1 の通り、内側の j に関するループにより $\sum_{j=1}^{64} M_{i,j} \times x_j$–(a) の積和計算を行い、外側の i に関するループにより (a) の結果を s_i に代入している。3.2 節において述べた通り、内側を i に関するループ、外側を j に関するループとするほうが、内側ループの入力パターン数を削減できるため、ループの構造を変更する。すなわち、内側の i に関するのループでは $s_i = s_i + M_{i,j} \times x_j \quad (i = 1, \cdots, 32)$–(b) を計算し、外側ループでは $j = 1, \cdots, 64$ に関する計算を行う。さらに、内側ループ全体を再利用するために、関数として切り出す。このとき、切り出した関数内において (b) を計算すると、s_i も入力となるため、再利用の可能性が低くなる。したがって、関数として切り出すのは $M_{i,j} \times x_j \quad (i = 1, \cdots, 32)$ の部分のみとする。すなわち、切り出した関数内の i ループにおいて $t_i = M_{i,j} \times x_j$ を計算し、関数の外側では $s_i = s_i + t_i \quad (i = 1, \cdots, 32)$ を計算する。もちろん、s_i は j のループに入る前に、0 に初期化しておく。このように切り出した関数の入力は x_j の値および定数 $M_{i,j} \quad (i = 1, \cdots, 32)$、また、出力は t_i となり、再利用の可能性を高めることができる。以上の再利用適用手法を (S)、切り出した関数を FUNC(S) と呼ぶことにする。(S) によるオーバーヘッドは、関数呼び出しのオーバーヘッド、s_i の初期化、t_i への一時退避および $s_i = s_i + t_i \quad (i = 1, \cdots, 32)$ のループ制御の 4 点である。

FUNC(S) における入力アドレス数を見ると、$M_{i,j}$ が 2048ヶ所の主記憶アドレスを参照している。$M_{i,j}$ は定数であるため、入力のパターン数に与える影響は少ないものの、RF の READ エントリに登録し切れない可能性がある。そこで、(S) に加えて、$M_{i,j}$ のバッファリングを行った。すなわち、切り出した関数に入る前に、$M_i^* = M_{i,j}$ の移しかえを行い、M^* を FUNC(S) の入力として用いる。この手法を (S+buf) と呼ぶことにする。(S+buf) によるオーバーヘッドは、(S) のオーバーヘッドおよび $M_i^* = M_{i,j}$ の移しかえである。

さらに、FUNC(S) への曖昧再利用の適用を考える。倍精度浮動小数点型により宣言されている FUNC(S) の定数でない入力 x_i を単精度浮動小数点型にキャストした上で、下位 20bit を無視した。この曖昧化手法を (S+tol) および (S+buf+tol) と呼ぶことにする。

4.3 MDCT への適用手法

3.3 節であげた、式 2 の部分について関数切り出しを行う。bladeenc では、式 2 の通り、内側の j に関するのループでは $\sum_{j=1}^{36} W_j \times x_j \times C_{ij}$–(c) の積和計算を行い、外側の i に関するのループでは (c) の結果を s_i に代入している。3.3 節において述べた通り、内側を i に関するのループ、外側を j に関するのループとするほうが内側ループの入力パターン数が減少するため、このように変更する。すなわち、内側の i に関するのループでは $s_i = s_i + W_j \times x_j \times C_{i,j}$–(d) を計算し、これを外側のループにおいて $j = 1, \cdots, 36$ について行う。さらに、内側のループ全体を再利用するために、関数として切り出す。このとき、切り出した関数内において (d) を計算すると、s_i も入力となるため、入力パターン数が増え、再利用の可能性が低くなる。したがって、関数として切り出すのは $W_j \times x_j \times C_{i,j} \quad (i = 1, \cdots, 18)$ の部分のみとする。つまり、切り出した関数内の i ループでにおいて $t_i = W_j \times x_j \times C_{i,j}$ を計算し、関数の外側では $s_i = s_i + t_i \quad (i = 1, \cdots, 36)$ を計算する。もちろん、s_i は j のループに入る前に、0 に初期化しておく。このように切り出した関数の入力は x_j の値および定数 W_j と $C_{i,j} \quad (i = 1, \cdots, 18)$、また、出力は t_i となり、再利用の可能性を高めることができる。以上の再利用適用手法を (M)、切り出した関数を FUNC(M) と呼ぶことにする。(M) によるオーバーヘッドは、関数呼び出しのオーバーヘッド、s_i の初期化、t_i への一時退避、および $s_i = s_i + t_i \quad (i = 1, \cdots, 36)$ のループ制御の 4 点である。

前節において行った定数群のバッファリングは、ここでは行わない。定数 W_j および $C_{i,j}$ の個数が前項の $W_{i,j}$ に比べて少なく、また、入力のバリエーションに影響を与える j が 1 から 36 までしかないことから、バッファリングを行わなくても再利用の可能性は十分高いと考え、逆にオーバーヘッドがかさむことを避けるためである。

さらに、FUNC(M) への曖昧再利用の適用を行う。FUNC(M) の定数でない入力である x_i はサブバンドフィルターバンクの出力であり、倍精度浮動小数点型により宣言されているため、これを単精度浮動小数点型にキャストした上で、下位 23bit を無視した。この曖昧化手法を (M+tol) と呼ぶことにする。

4.4 量子化処理への適用手法

3.4 項であげた、式 3 の関数切り出しを行う。bladeecnd では、式 3 は量子化テーブルを参照することにより高速化されている。量子化テーブルの参照を行う関数では、$x \times m$–(e) の値を入力としている。(e) の計算は関数内で実行したほうが再利用されたときの削減命令数が多くなるため、(e) の関数への埋め込みを行った。この手法を (Qt)、埋め込みの行われた関数を FUNC(Qt) と呼ぶことにする。FUNC(Qt) における入力は x および m である。

また、量子化テーブルを使用せずに、再利用の適用により高速化を図るために、実際に式 3 の計算を実行する関数を構成した。この手法を (Qp)、構成した関数を FUNC(Qp) と呼ぶことにする。FUNC(Qp) における入力は、FUNC(Qt) と同様、x および m である。(Qt) および (Qp) により発生するオーバーヘッドは、関数呼び出しのオーバーヘッドである。

さらに、FUNC(Qt) および FUNC(Qp) への曖昧再利用の適用を行う。FUNC(Qt) および FUNC(Qp) の入力である x は MDCT の出力であり、倍精度浮動小数点型で宣言されているため、これを単精度浮動小数点型にキャストした上で、下位 23bit を無視した。この手法を (Qt+tol) および (Qp+tol) と呼ぶことにする。

一方、量子化テーブルを使用せずに、再利用の適用により高速化を図る場合、$X^{0.75}$–(f) の計算への曖昧再利用が考えられる。式 3 においては、(f) の出力が整数に丸められるため、X を大幅に曖昧化した場合でも、誤差の影響は無視できると考え、倍精度浮動小数点数型で宣言されている X を整数型にキャストすることにした。この曖昧化手法を (P+tol) と呼ぶことにする。

4.5 符号化ビット数計算への適用手法

3.4 節であげた符号化ビット数計算において、表参照を行う命令区間を、2 つの整数値を入力とする関数として切り出す。この手法を (H)、切り出した関数を FUNC(H) と呼ぶことにする。FUNC(H) における 2 つの入力値は Huffman テーブル参照のインデックスとなるため、曖昧再利用は適用できない。また、この関数切り出しによるオーバーヘッドは、関数呼び出しのオーバーヘッドのみである。

FUNC(H) における 2 つの入力値は、20 以下であることがほとんどなので、FUNC(H) における入力パターン数は少なく、また、切り出した命令区間も小

さいため、アンローリングの効果が期待できると考える。ここでは、2 重アンローリングを施し、4 つの整数を入力とし、Huffman テーブルへの参照を 2 回行う命令区間を関数として切り出した。この手法を (H+ur)、切り出した関数を FUNC(H+ur) と呼ぶことにする。

4.6 量子化誤差計算への適用手法

3.4 節であげた、式 4 について関数切り出しを行う。x,y,s を入力とし、式 4 の計算結果が出力となる命令区間を関数として切り出す。この手法を (N)、切り出した関数を FUCN(N) と呼ぶことにする。(N) により発生するオーバーヘッドは、関数コールのオーバーヘッドである。FUNC(N) における入力は 3 個だけであり、また、切り出した命令区間は非常に小さいので、アンローリングをによるオーバーヘッドの削減を考える。連続して実行される式 4 においては、s は変化しないため、2 重アンローリングを施した場合、$x1$,$y1$,$x2$,$y2$,s の 5 個を入力とする関数に切り出すことになる。この手法を (N+ur)、切り出した関数を FUNC(N+ur) と呼ぶことにする。

式 4 の計算において、s は量子化ステップより計算される値であり、y は 8206 以下の整数である。実際には、y のほとんどが 100 以下となっているので、入力パターン数を増やしているのは x であると言える。また、x は y および s ほど演算結果をに影響を与えないため、曖昧化することができる。x は MDCT の出力であり、倍精度浮動小数点型で宣言されているため、単精度浮動小数点型にキャストした上で、下位 23bit を無視する曖昧化を行った。この曖昧化手法を (N+tol) および (N+ur+tol) と呼ぶことにする。

さて、bladeenc では、式 4 中の $y^{4/3}$–(g) の部分を、プログラムの最初に作成した表を参照することにより高速化している。3.4 節において、この表参照は再利用により置き換えることができると述べた。すなわち、プログラムの最初において $y^{4/3}$ の表の作成は行わず、また、表参照を行っている部分を $y^{4/3}$ に書き換える。この手法を (T) と呼ぶことにする。

第 5 章　性能評価

評価には、再利用機構を搭載した単命令発行の SPARC-V8 アーキテクチャ・シミュレータを用いた。シミュレータの各パラメータを表 3 に示す。測定対象は、

表 3: シミュレーション時のパラメータ

D-Cache 容量	64 Kbytes
ラインサイズ	64 bytes
ウェイ数	4
Cache ミスペナルティ	20 cycles
Register-Window	6 sets
Window ミスペナルティ	20 cycles/set
ロードレイテンシ	2 cycles
整数乗算レイテンシ	8 cycles
整数除算レイテンシ	70 cycles
浮動小数点加減乗算レイテンシ	4 cycles
単精度浮動小数点除算レイテンシ	16 cycles
倍精度浮動小数点除算レイテンシ	19 cycles
RW の最大深さ	4
RB(レジスタ) - Register 比較	1 cycle
RB(READ) - Cache 比較	4 bytes/cycle
RW(WRITE) → Cache 書き込み	4 bytes/cycle
RB(レジスタ) → Register 書き込み	1 cycle

表 4: 評価対象となる再利用適用手法

	手法	ベース	内容	表 2 対応
	(B)		無修正の bladeenc(量子化テーブル使用)	
	(P)		量子化テーブルを用いずに式 3 を実行	
聴覚心理モデル	(A+tol)	(B) or (P)	$tan^{-1}y/x$ の曖昧化 (下位 20bit 無視	3
サブバンド	(S)	(B) or (P)	i に関する $M_{i,j}x_j$ のループの関数切り出し	5
フィルターバンク	(S+buf)	(B) or (P)	(S) + M のバッファリング	6
	(S+tol)	(B) or (P)	(S) + 曖昧化 (下位 20bit 無視)	6
	(S+buf+tol)	(B) or (P)	(S) + M のバッファリング + 曖昧化 (下位 20bit 無視)	6
MDCT	(M)	(B) or (P)	i に関する $W_j x_j C_{i,j}$ のループの関数切り出し	9
	(M+tol)	(B) or (P)	(M) + 曖昧化 (下位 23bit 無視)	9
量子化ループ	(Qt)	(B)	量子化テーブル参照の関数切り出し	11
	(Qt+tol)	(B)	(Qt) + 曖昧化 (下位 23bit 無視)	11
	(Qp)	(P)	式 3 の関数切り出し	11
	(Qp+tol)	(P)	(Qp) + 曖昧化 (下位 20bit 無視)	11
	(P+tol)	(P)	$X^{0.75}$ の曖昧化 (浮動小数点を整数化)	11
	(H)	(B) or (P)	Huffman テーブル参照を行う関数の切り出し	12
	(H+ur)	(B) or (P)	(H) + 2 重アンローリング	12
	(N)	(B) or (P)	式 4 の関数切り出し	13
	(N+ur)	(B) or (P)	(N) + 2 重アンローリング	13
	(N+tol)	(B) or (P)	(N) + 曖昧化 (下位 32bit 無視)	13
	(N+ur+tol)	(B) or (P)	(N) + 2 重アンローリング + 曖昧化 (下位 32bit 無視)	13
	(T)	(B) or (P)	初期累乗計算の省略	13

ソース無修正の bladeenc、bladeenc において量子化テーブルを用ずに式 3 を実行するもの、および前章で述べた各再利用適用手法を施したものを gcc-3.0.2(-msupersparc -O2) によりコンパイルし、スタティックリンクにより生成したロードモジュールである。なお、各再利用手法を適用したものと対応させるために、無修正の bladeenc を (B)、量子化テーブルを用いずに実行するものを (P) と呼ぶことにする。

また、測定に用いる音声/音楽データとして、CLASSIC, POPS, VOICE の 3 種類を用意した。

5.1 単独の手法ごとの評価

前章で示した bladeenc の各手法の評価を行う。評価対象となる手法を表 4 にまとめて示し、また、各手法の表 2 との対応関係についても示した。それぞれの手法を単独に適用したプログラムについて、再利用機構上でのシミュレーションを行い、再利用の効果を測定する。ここでは、RF エントリ数、RB エントリ

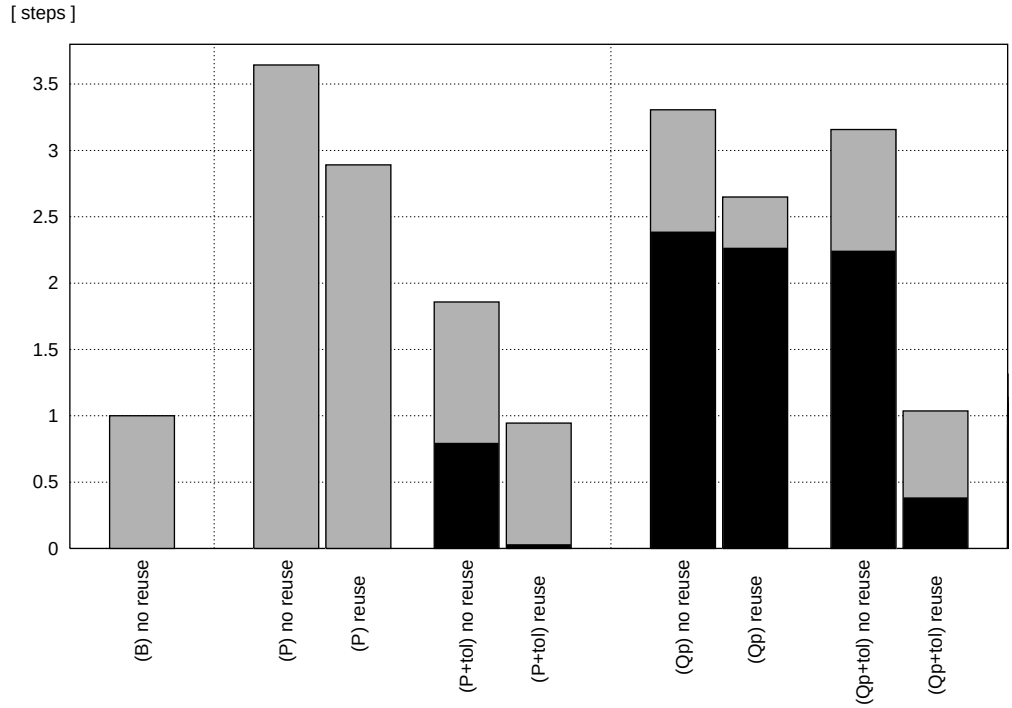
数、READ および WRITE のエントリ数が、それぞれ、32,1024,1024,256 である再利用表を用いた。ただし、(S) においては、2048 個の定数を参照しているため、READ のエントリ数を 2048 とした。また、入力音声については POPS を使用した。

各手法をプログラムに適用したものは、ソースコードの書き換えにおけるベースとして、(B) を基準としたものと (P) を基準としたものに分けられる。ここでは、(Qp), (Qp+tol) および (P+tol) は (P) を基準とし、それ以外の手法は (B) を基準としている。

まず、(P) を基準として再利用を適用したものについて、評価を行う。(Qp), (Qp+tol), (P+tol) および (P) の手法を適用した各プログラムに対して、再利用を適用したとき、および適用しなかったときの実行命令ステップ数を図 3 に示す。なお、各棒グラフにおいて黒塗りで示されているのは、各手法において、再利用の対象とするために、切り出しおよび入力の曖昧化を適用した関数、すなわち FUNC(Qp) および $X^{0.75}$ の命令実行ステップである。それぞれが非常に効果的に再利用され、3 倍以上の命令ステップ数削減に成功していることが確認できる。

次に (B) を基準として各手法を適用したプログラムについても同様に、再利用を適用したもの、およびしなかったものにおける実行命令ステップ数を図 4 に示す。ここでも、各手法において、再利用の対象とするために、切り出しおよび入力の曖昧化を適用した関数の命令実行ステップを黒塗りで示している。すなわち、それぞれの手法における、$tan^{-1}y/x$, FUNC(S), FUNC(S+buf), FUNC(M), FUNC(H), FUNC(H+ur), FUNC(N), FUNC(N+ur), $y^{4/3}$ の命令ステップである。いずれの変更点においても、曖昧再利用により、bladeenc に変更を加えずに再利用なしで実行した場合の実行命令ステップ数より少ない命令ステップ数で実行できることが分かる。特に、(S),(M) 以外の変更を施した場合においては、無修正の bladeenc に対して再利用を適用したときよりも高速化されている。しかしながら、(H) および (N) の変更点においては、切り出した命令区間の再利用が全体の命令ステップ数削減に効果を及ぼしているわけではなく、MDCT の出力を曖昧化することにより、量子化ループにおける処理が全体的に再利用されやすくなったようである。また、(H+ur) および (N+ur) におけるアンローリングには、ほとんど効果がないことが分かる。

ところで、図 3,4 のどちらにおいても、(B) において再利用を適用せずに実行



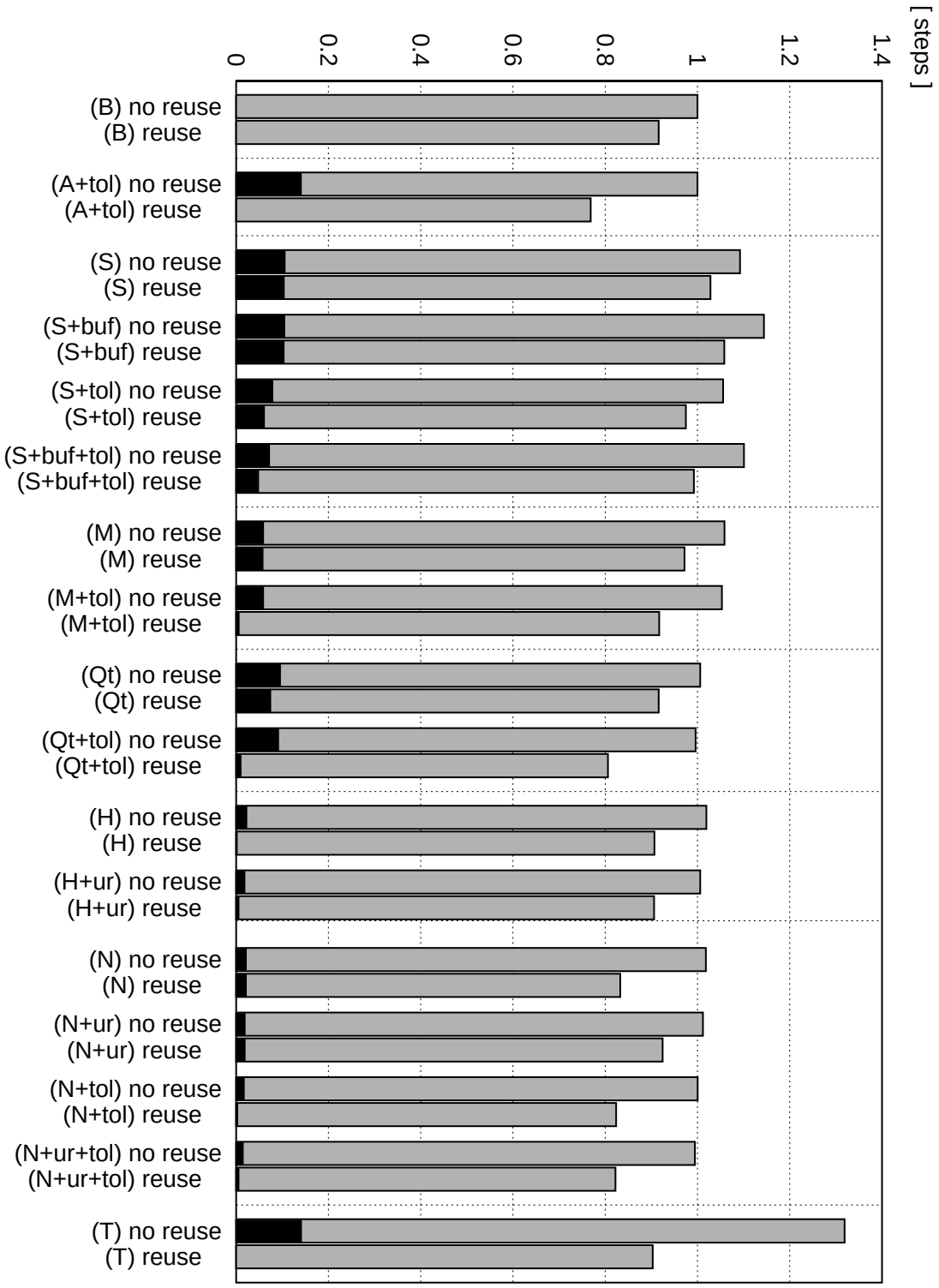
命令ステップ数による比較 – (B) 再利用なしにおける命令ステップ数を 1 として正規化 –
黒塗部分は各手法において切り出した関数の実行ステップ数
RF = 32, RB = 1024, READ/WRITE = 1024/256
入力音声 : POPS

図 3: (P) を基準とした各手法の単独での効果

した場合の実行命令ステップ数を 1 としている。つまり、(Qp+tol) および (P+tol) の曖昧再利用により、量子化テーブルを用いなくても、式 3 および $X^{0.75}$ における計算を十分に高速化することができることが分かる。bladeenc での量子化テーブルは人為的に作成したものであったが、再利用機構では計算機が自動的に表を作成できるため、プログラマの手間を省くことができる。

5.2 複数の手法を組み合わせた評価

前節の結果をふまえて、各手法を組み合わせて適用したプログラムに対して再利用の適用を試みる。個別の評価において結果の良くなかった (S) および (M) については変更点から除き、(1): (A+tol) (Qt+tol) (H+ur+tol) (N+ur+tol) (T); (2): (A+tol) (Qp+tol) (P) (C+ur+tol) (N+ur+tol) (T); の組み合わせとした。つまり、(1) が (B) を基準とした再利用適用プログラム、(2) が (P) を基準とした再利用適用プログラムである。前節では、(A), (H), (N), (T) は、(B) を基準とした手法であると述べたが、実際は量子化テーブルとは関係のない部分なので、(P) を基



命令ステップ数による比較 – (B) 再利用なしにおける命令ステップ数を 1 として正規化 –
黒塗部分は各手法において切り出した関数の実行ステップ数
RF = 32, RB = 1024, READ/WRITE = 1024/256
入力音声 : POPS

図 4: (B) を基準とした各手法の単独での効果

準としても実現できる手法である。また、RF, RB, READ/WRITE のエントリ数を、それぞれ、(a): 32, 512, 128/4; (b): 32, 1024, 128/4; (c): 32, 1024, 1024/256; と変化させて測定した。さらに、再利用のハードウェアも考慮に入れた評価をするために、命令ステップ数および再利用機構のオーバーヘッドを含めた全実行サイクル数により測定した。再利用機構のオーバーヘッドとは、表 3 で示した RB(レジスタ) - Register 比較および RB(READ) - Cache 比較、RB(レジスタ) → Register 書き込みおよび RW(WRITE) → Cache 書き込み、キャッシュミス、レジスタウィンドウミスにおけるサイクル数である。また、入力の違いによる影響を見るために、CLASSIC, POPS, VOICE のそれぞれを入力とした測定を行った。

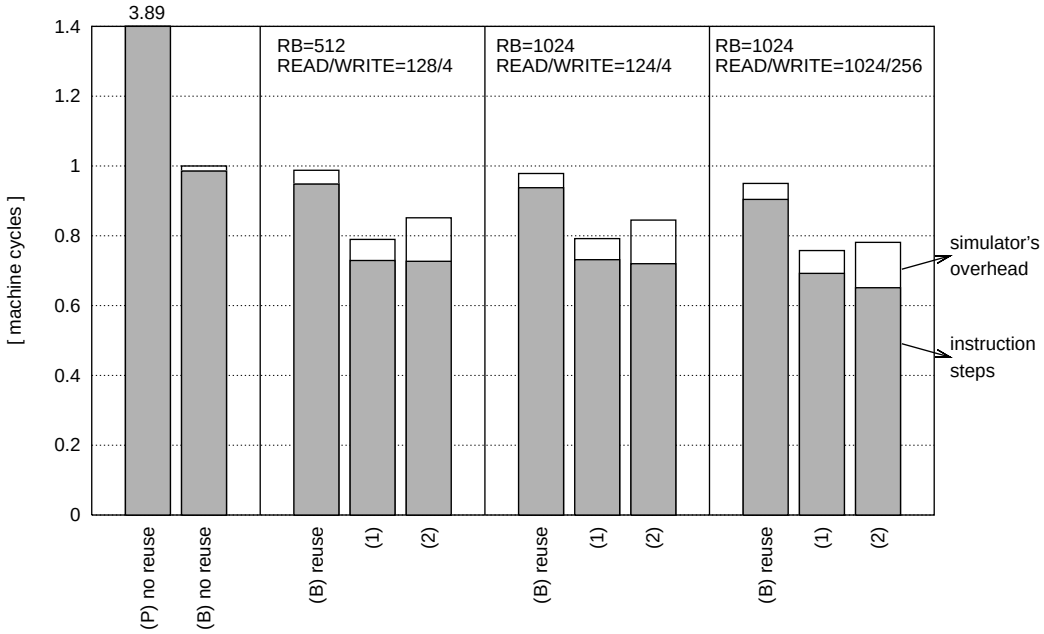
測定の結果を図 5,6, 7 に示す。(1) においては、(B) を再利用なしで実行した場合と比べて 20 ~ 25%程度のサイクル数削減に成功している。また、(2) において、(P) を再利用なしで実行した場合と比べて約 4 倍の高速化を実現し、さらに、量子化テーブルを使用していないにもかかわらず、(B) に再利用を適用した場合に比べて約 20%程度の高速化に成功している。

また、再利用表のエントリ数の増加により、測定結果は多少向上したが、(c) は (a) の約 20 倍の大きさであるにもかかわらず、性能はわずかに 5%ほど上昇したのみであった。(a) の再利用表における RF1 エントリあたりの RB の大きさは、量子化テーブルと同程度であり、量子化テーブルを RF1 エントリで代用した上で、他の RF エントリにより、高速化を実現していると考えることができる。

また、(2) における命令ステップ削減数は、わずかに (1) を凌駕している。にもかかわらず、再利用機構のオーバーヘッドにより、マシンサイクルレベルの高速化では (1) に劣る。これは、式 3 における累乗計算を実行する際に、RB に多数のエントリが登録され、入力の照合においてオーバーヘッドが増加するためである。表 3 で示したように、RB の READ の値と Cache の値の比較は、1 サイクルで 4byte となっている。再利用機構にけるエントリ比較の高速化が実現できれば、(2) のようにエントリ比較が実行速度に大きな影響を与えている場合を解決できる。

第6章　まとめ

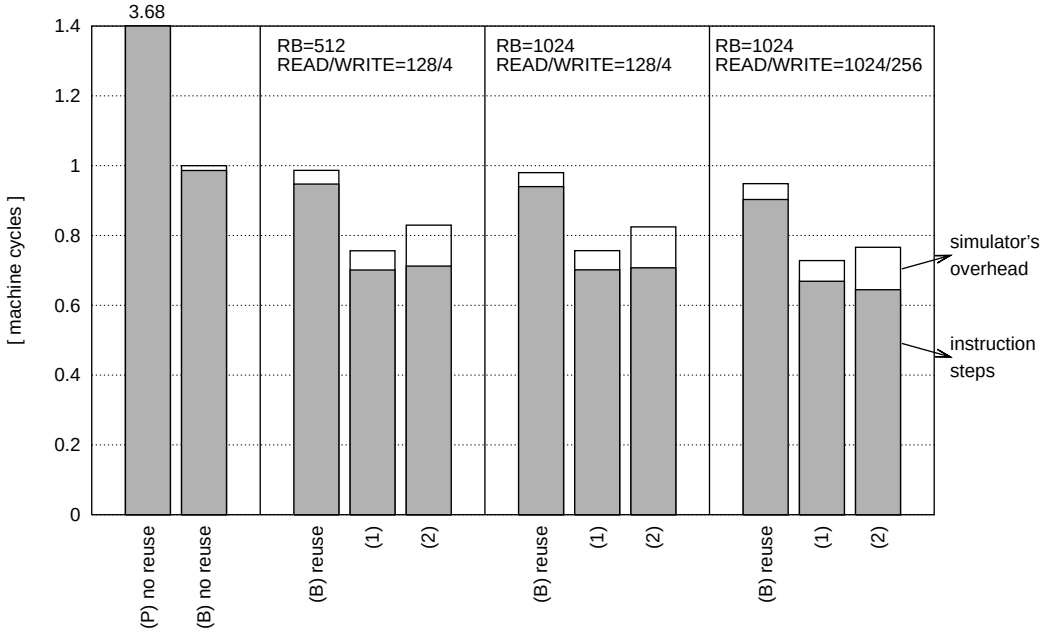
本研究では、MP3 エンコーダに対して、プログラムのソースコードを書き換えることにより曖昧再利用を適用し、高速化を図る手法を提案した。再利用可



(1): (A+tol) (Qt+tol) (H+ur+tol) (N+ur+tol) (T)

(2): (A+tol) (Qp+tol) (P) (H+ur+tol) (N+ur+tol) (T)

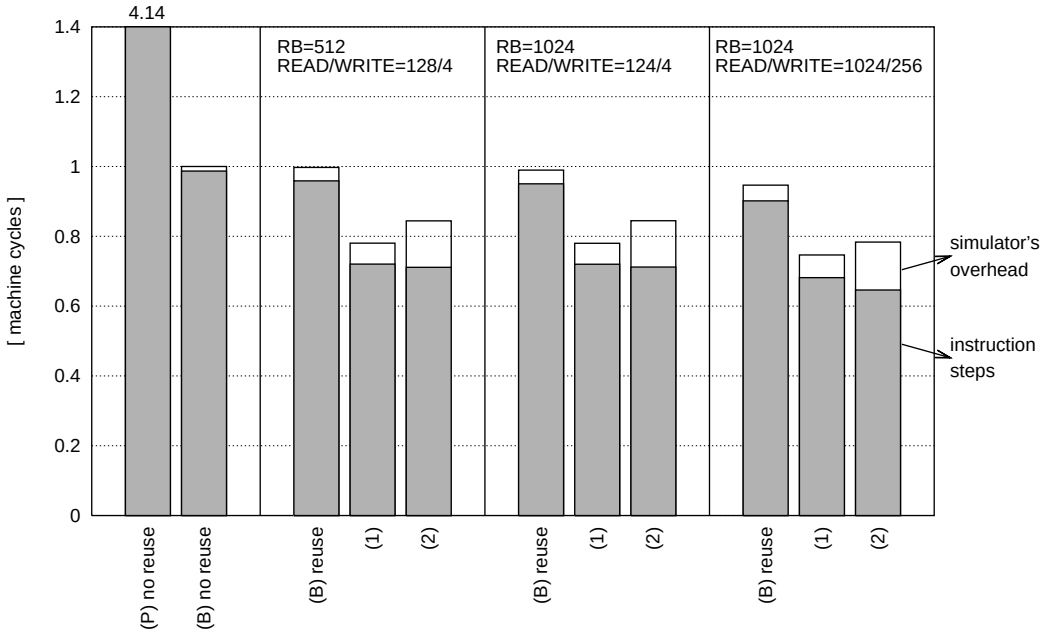
図 5: 組み合わせた手法におけるサイクル数による評価–CLASSIC–



(1): (A+tol) (Qt+tol) (H+ur+tol) (N+ur+tol) (T)

(2): (A+tol) (Qp+tol) (P) (H+ur+tol) (N+ur+tol) (T)

図 6: 組み合わせた手法におけるサイクル数による評価–POPS–



(1): (A+tol) (Qt+tol) (H+ur+tol) (N+ur+tol) (T)
(2): (A+tol) (Qp+tol) (P) (H+ur+tol) (N+ur+tol) (T)

図 7: 組み合わせた手法におけるサイクル数による評価–VOICE–

能性が高いと思われる命令区間を関数に切り出すことにより、再利用の適用可能な状態にした上で、命令区間における入力値を曖昧化し、再利用率の向上による命令削減、高速化を実現する。MP3 エンコーダに対し様々な再利用適用手法を施した結果、量子化テーブルを使用しない場合においては約 4 倍、使用した場合と比べても 20 ~ 25%程度の命令サイクル数削減に成功した。

ところで、本研究で示した曖昧再利用の効果は音声入力データの局所的類似性によるものではない。MP3 エンコーダの各処理プロセスにおける入力データ数が多いためであると思われる。(S) および (M) においては入力データの一部を切り出して処理しており、入力の局所類似性の効果が大きいと思われる命令区間であるが、オーバーヘッドにより再利用効果が消されている。より少ないオーバーヘッドで局所類似性の効果が大きい区間を切り出すことができれば、さらなる高速化を図ることができる。

謝辞

本研究の機会を与えてくださった、富田眞治教授に深く感謝の意を表します。

また、本研究に関して適切なご指導を賜った中島康彦助教授、森眞一郎助教授、五島正裕助手、津邑公暁助手に深く感謝いたします。

さらに、日頃暖かく御鞭撻下さった京都大学工学部情報学科富田研究室の諸兄に心より感謝いたします。

参考文献

[1] J.Yang and R.Gupta.「Load Redundancy Removal through Instruction Reuse」. ICPP, 2000.

[2] J.Yang and R.Gupta.「Energy-efficient load and store reuse. ISLPED」, pp.72-75, 2001.

[3] 中島康彦, 緒方勝也, 正西申悟, 五島正裕, 森眞一郎, 北村俊明, 富田眞治:「関数値再利用および並列事前実行による高速化技術」, 情報処理学会論文誌: ハイパフォーマンスコンピューティングシステム, HPS5,pp.1-12,Sep.2002.

[4] Carlos Alvarez jesus Corbal, Esther Salami, Mateo Valero: 「On the Potential of Tolerant Region Reuse for Multimedia Applications」

[5] R.P.Paul. 「SPARC Architecture, Assembly Language Programming, and C. Prentice-Hall」, 1999.

[6] 浦田敏道 「詳細 MP3 マニュアル」エム研, 1999.12.