計算機アーキテクチャ 152－25
（２００３．３．１１）

曖昧再利用によるMP3エンコーダの高速化手法

竹村尚大[†] 津邑公暁[†] 中島康彦[††]
五島正裕[†] 森眞一郎[†] 富田眞治[†]

コンパイラによる専用命令の追加を必要としない区間再利用機構を前提とした上で，MP3 エンコーダに対して区間再利用の適用可能性を検討し，高速化を行った。また，入力値に許容範囲を設ける曖昧再利用を適用することで，更なる高速化を図った。エンコーダを分析のうえ，各処理部を対象とした合計 20 種類の再利用適用手法を提案し，それぞれの効果について評価を行った。手法を単独で適用した場合，再利用を全く適用しないものと比して最大 25%の命令ステップ数を削減することができた。また，効果のあった手法を複数組み合わせて適用した結果，再利用処理のオーバヘッドを含めても最大で約 30%のサイクル数を削減することができた。

A Technique to Speedup MP3 Encoding with Tolerant Reuse

Naohiro Takemura,[†] Tomoaki Tsumura,[†] Yasuhiko Nakashima,[††]
Masahiro Goshima,[†] Shin-ichiro Mori[†] and Shinji Tomita[†]

This paper describes how to apply tolerant reuse against MP3 audio encoder, assuming the hardware which can reuse the region automatically without any special instructions controlled by compiler. We analized MP3 encoder deeply, and investigated the probability of region reuse and how to apply region/tolerant reuse on MP3 encoder. We designed 20 methods to apply region/tolerant reuse against MP3 encoder. In the evaluation of the case using single method, 25% instruction steps in the maximam could be reduced. In the case using multiple method, we found that 30% instruction cycles can be redused in the maximam.

1. 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，それらのデータサイズは本質的に大きく，圧縮/伸張が不可欠である。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，それを高速に行う技術が必要となる。

一方，高速化技術として，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，再度同一入力により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。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，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例として，JPEG エンコーダを用いた報告がある [4)]。通常の区間再利用に加えて，入力の一致判別に寛容性を持たせる曖昧再利用が提案されている。入力のバリエーションを減らすことで，同じ入力値の出現確率を高め，最終的に再利用率が向上する。

本稿では，専用命令を必要としない区間再利用機構を前提とし，音声圧縮技術 MPEG Audio Layer 3 (以下 MP3[5)]) のエンコーダに曖昧再利用を適用することで高速化を図る。MP3 は非可逆圧縮技術であるため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あるため，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，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と考える。

2. 曖昧再利用技術

本章では，曖昧再利用の基礎的技術である区間再利用 (以下，再利用と略す) について概要を述べる。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。区間の開始/終了を知らせる専用命令をコンパイラに埋め込ませることによってハードウェアに識別させる方法もあるが，本稿では SPARC ABI[6)] の規定に基づき区間を自動的に識別して再利用を行う機構 [3)] を仮定する。この機構は，既存のロード

† 京都大学
Kyoto University
†† 京都大学 / 科学技術振興事業団さきがけ研究 21
Kyoto University / PRESTO, JST

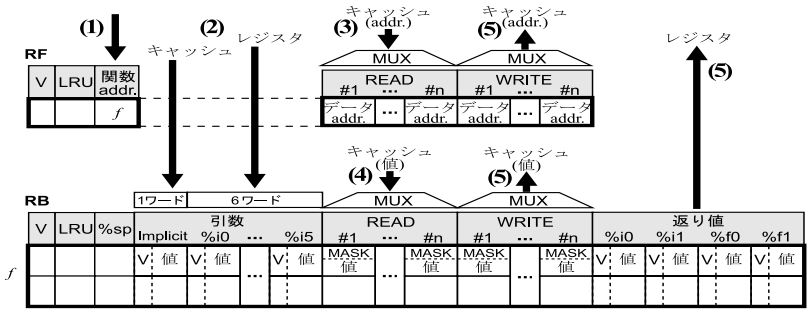

図 1 再利用表の構成

モジュールをそのまま利用できるという利点がある。今回適用するのは，関数呼出に対する区間再利用 (以下，関数再利用と呼ぶ) である。

2.1 再利用機構とその動作

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。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。各表の V と LRU は，有効エントリの表示およびエントリ入換えのヒントである。RF は，関数の先頭アドレス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，引数，主記憶値，返り値を保持する。返り値は，%i0〜1 (リーフ関数では%o0〜1) または%f0〜1 に格納される。READ アドレスは RF，マスクおよび値は RB が管理することにより，READ アドレスと RB の複数エントリを CAM により一度に比較する構成が可能となる。

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。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。その後，復帰命令を実行した時点で登録中のエントリを有効にする。ただし復帰までに，他の関数の呼出，入出力数の容量オーバ，システムコールや割込の発生，などの擾乱が生じた場合はその時点で登録を打ち切る。

以後は，関数を呼び出す前に，(1) 関数先頭アドレスを検索し，(2) 引数が完全一致するエントリを選択し，(3) 関連する主記憶 READ アドレスを全て参照して，(4) 一致比較を行う。全ての入力が一致した場合，(5) 登録済の出力 (返り値，大域変数，および親関数の局所変数) を書き戻すことで，関数の実行を省略できる。

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)** 予想される再利用関数の数; に留意しなければならい。

(1) 各 RF エントリに登録された主記憶アドレスも一致比較の対象となるため，同一関数であっても，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異なる RB エントリに記憶される。よって，参照する主記憶アドレスの数が登録可能な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。このような状況を防ぐために，再利用を期待する関数では，参照する主記憶アドレス数に留意する必要がある。

(2) 各 RF に割り当てられた RB エントリを全て使い切ると，さらに RB エントリに登録するためには，不要と思われる RB エントリを追い出す必要がある。入力パターン数が RB の最大エントリ数より多いと，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，関数の大きさ，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。

(3) 大規模なプログラムを実行する場合，一般に多くの関数が出現する。関数の数が RF の最大エントリ数を越えた場合，古い RF エントリが追い出される。一般に，RF に登録されている関数の中から，再利用効率が最も悪い関数を厳密にハードウェアが特定することは困難であり，関数が多数検出されると，将来有効に再利用されるはずの関数が 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。本稿では，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関数が常に RF に存在するよう工夫した。

2.3 プログラミング手法

本節では，実際にソースコードに適用するための具体的手法を述べる。なお各手法は，いずれも再利用による命令数削減とソースコード書き換えによるオーバ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。

再利用区間の切り出し: 再利用機構が特定できる命令区間は関数であるため，再利用を適用すべき区間を関数に切り出す。この際には，関数内の入出力アドレス数が RF の READ/WRITE エントリ数を著しく越えないように調整する必要がある。

バッファリング: 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，関数 C において，大きさ n の配列 a のうち $m(m < n)$ 個が参照され，C の各繰り返しが参照する m 個が互いに異なる主記憶アドレスとする。その場合，C が参照する m 個の値自身が全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，C の各繰り返しの直前において，C が参照する m 個の値を，一旦大きさ 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。C が参照するアドレスは b の 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。以下この方法を**バッファリング**と呼ぶ。出力先の主記憶アドレスが変化する関数に対しても，同様にバッファリングを適用することができる。バッファリングは前述した (1) および (2) の留意点に対して有効な手法である。関数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 アドレスエントリの溢れを防ぐことができる。また，a の取り得る値を α 個とすると，入力パターン数 $\alpha^{n/m} \times m$ は，バッファリングにより $\alpha^{n/m}$ まで減少し，RB エントリの溢れを防ぐことができる。

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関数において，ループ制御や関数呼び出しのオーバヘッドを削減する方法に，アンローリングがある。関数の繰り返しを一つにまとめ，新たな関数とする。r 回を一つにまとめると，入力パターン数は最大 r 乗に増加するものの，関数呼出のオーバヘッドは約 $1/r$ 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，かつ当該関数の実行回数が多く，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，より高速化することができる。小さい関数ほどオーバヘッドの占める割合が高く，アンローリングの効果は高い。

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。

3. MP3 エンコーダへの適用

本章では，MP3 エンコーダの各部への再利用の適用について詳述する。対象として bladeenc ver0.92.0((c) Copyright 1998, 1999 - Tord Jansson) を用いた。

MP3 エンコーダは，一定の周波数で音声をサンプリングした PCM 信号を圧縮する。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; からなる。bladeenc における各処理時間の割合を**表 1** に示す。各処理は，1152 個のサンプリングデータからなる 1 フレーム，または 576 個からなる 1 グラニュールという単位の PCM 信号を入力とする。

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

聴覚心理を考慮した計算により，MDCT の窓幅や，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。その内部では，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。FFT の内部のバタフライ演算は 6 個の実数，$tan^{-1}y/x$ についても 2 個の実数を入力とするため，再利用可能性は高い。

今回は $tan^{-1}y/x$ を計算する関数に対して，入力 x および y の下位 20bit を無視する曖昧再利用を適用することにした。この手法を **(A+tol)** とする。

3.2 サブバンドフィルタバンク

デジタルフィルタにより入力信号を 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。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バンク全体の処理時間の約 80%を占めている。なお，$M_{i,j}$ は定数，x_j は実数入力値である。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，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。

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

このためループを入れかえ，内側ループでは $s_i = s_i + M_{i,j} \times x_j \quad (i = 1, \cdots, 32)$ を計算し，外側ループではこれを $j = 1, \cdots, 64$ に対して行う。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 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。以上の再利用適用手法を **(S)**，切り出した関数を **FUNC(S)** とする。(S) によるオーバヘッドは，関数呼び出し，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) では，$M_{i,j}$ が 2048 個の入力アドレスを参照しており，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う。この手法を **(S+buf)** とする。

さらに，FUNC(S) への曖昧再利用の適用を考える。double 型である FUNC(S) の入力 x_i を float 型にキャストした上で，下位 20bit を無視する。この手法をそ

れぞれ **(S+tol)** および **(S+buf+tol)** とする。

3.3 MDCT

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。MDCT の内部について調査すると，式 2 に示す計算が処理時間の 90%以上を占めている。ただし，W_j,$C_{i,j}$ は定数，X_j は実数入力値である。ここで，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。これにより入力は x_j のみとなり，また，$j = 1,\cdots,36$ であることから，ここで用いられる定数は 36 組となり，再利用可能性が高まる。

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

よって前節同様ループを入れかえ，内側ループでは $s_i = s_i + W_j \times x_j \times C_{i,j}$ を計算し，これを外側ループにおいて $j = 1,\cdots,36$ 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，前節と同じ理由から $W_j \times x_j \times C_{i,j}$ $(i = 1,\cdots,18)$ の部分のみを関数として切り出す。以上の再利用適用手法を **(M)**，切り出した関数を **FUNC(M)** とする。(M) によるオーバヘッドは，関数呼び出し，s_i の初期化，t_i への一時退避，および $s_i = s_i + t_i$ $(i = 1,\cdots,36)$ のループ制御の 4 点である。

なお，前節で行った定数群のバッファリングは，ここでは行わない。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 までしかないためである。

さらに，FUNC(M) に曖昧再利用を適用する。入力 x_i はサブバンドフィルタバンクの出力であり，double 型である。これを float 型にキャストした上で，下位 23bit を無視する。この曖昧化手法を **(M+tol)** とする。

3.4 量子化ループ

バタフライ演算の出力に対し，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。

量子化処理では，式 3 に示す計算が繰り返し実行される。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。

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

bladeenc では，$nint(X^{0.75} - 0.0946)$ における X および計算結果を格納する 16 万エントリの表 (以下，量子化表と呼ぶ) を用意することで，高速化を実現している。量子化表を使用しない場合，式 3 の計算に要する時間はエンコード処理全体の約 70% にも上る。量子化表の参照を行う関数は，$x \times m$ の値を入力としている。こ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，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)** とする。

また，量子化表を使用せずに，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)** とする。(Qt) および (Qp) により発生するオーバヘッドは，関数呼び出しのオーバヘッドである。

さらに，FUNC(Qt)，FUNC(Qp) に曖昧再利用を適用する。MDCT の出力で double 型であるこれらの入力 x を float 型にキャストした上，下位 23bit を無視する。この手法を **(Qt+tol)** および **(Qp+tol)** とする。

一方，量子化表を使用せずに，再利用の適用により高速化を図る場合，$X^{0.75}$ の計算への曖昧再利用も考えられる。$X^{0.75}$ の出力は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，float 型である X を int 型にキャストすることにした。この曖昧化手法を **(P+tol)** とする。

次に，符号化ビット数の計算では，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キーとして 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 未満の整数であるが，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。

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)** とする。FUNC(H) における 2 つの入力値は Huffman テーブル参照のキーとなるため，曖昧再利用は適用できない。この関数切り出しによるオーバヘッドは，関数呼び出しのオーバヘッドのみである。

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であるので，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。よって，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す。この手法を **(H+ur)**，切り出した関数を **FUNC(H+ur)** とする。

その他では，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。入力は，実数 x，x を量子化した値 y，量子化ステップ数 s の 3 つであり，再利用できる可能性が高い。bladeenc では，予め全ての y $(= 0,\cdots,8206)$ について $y^{4/3}$ を計算しておくことで，この部分の高速化を図っている。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，再利用でも高速化が期待できる。

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

式 4 を関数として切り出す手法を **(N)**，切り出した関数を **FUCN(N)** とする。FUNC(N) における入力は 3 個と少なく，また切り出した命令区間も非常に小さいので，アンローリングの適用を考える。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 x_1, y_1, x_2, y_2, s の 5 個

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 $W_{j}x_jC_{i,j}$ のループ	11 (3)	×
	9	i に関する $W_{j}x_jC_{i,j}$ 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)	◎

表 3　再利用適用手法

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

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，y は 8206 以下の整数である。ただ実際には y のほとんどは 100 以下の値で，入力パターン数を増やしているのは x である。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可能である。そこで，MDCT の出力で double 型であるこの x を float 型にキャストした上で，下位 23bit を無視する。この手法を **(N+tol)** および **(N+ur+tol)** とする。

さて bladeenc では，式 4 中の $y^{4/3}$ の部分を，予め作成しておいた表を参照することにより高速化している。この表参照を再利用に置きかえ，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)** とする。

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のうえ，各種適用手法を提案した。各処理における処理時間量と再利用可能性を**表 2** に，適用手法を**表 3** にまとめた。各処理時間は全体を 100 とした場合の値であり，量子化処理において量子化表を使用しない場合の値は括弧内に示した。

4. 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 シミュレータを用いた。各パラメータを**表 4** に示す。測定には，無修正の bladeenc **(B)**，量子化表を用ずに式 3 を実行するもの **(P)**，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を用いた。

表 4　シミュレーション時のパラメータ

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

4.1 単独の手法ごとの評価

表 3 で示した各手法の評価を行った。ここでは，RF, RB，READ/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 個の定数を参照するため，READ エントリ数を 2048 とした。なお (Qp), (Qp+tol) および (P+tol) は (P)，それ以外の手法は (B) のソースコードを基準とし，書換えを行っている。

まず，(P) を基準としたものについての評価結果を示す。(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，再利用の適用あり/なしの場合における実行命令ステップ数を**図 2** に示した。なお黒塗りで示した部分は，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。

次に (B) を基準としたものについて，再利用の適用あり/なしの場合における実行命令ステップ数を**図 3** に示した。ここでも同様に，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た。いずれの手法においても，再利用なしで無修正の bladeenc を実行した場合よりも，曖昧再利用を適用したものの方が良好な結果が得られた。特に (S), (M) 以外においては，無修正の bladeenc に対して再利用を適用したものよりも高速化されている。ただし，(H) および (N) においては，切り出した関数の再利用が全体の命令ステップ数削減に寄与しているわけではなく，MDCT の出力を曖昧化したことにより，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，ほとんど効果がなかった。

ところで，図 2, 図 3 のいずれにおいても，(B) において再利用を適用しない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，量子化表を用いずとも，式 3 および $X^{0.75}$ における計算を十分に高速化できることが分かる。bladeenc での量子化表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。

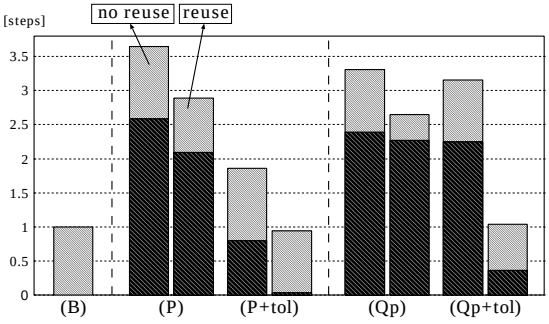

図 2 (P) を基準とした各手法の単独での効果

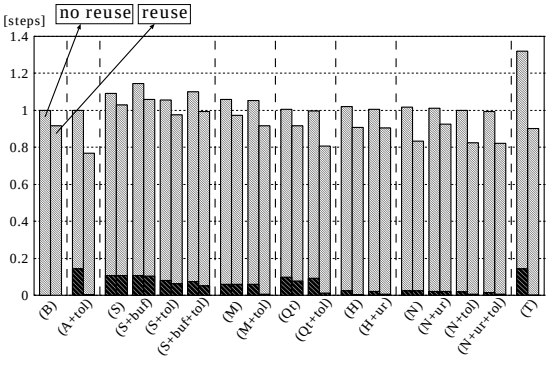

図 3 (B) を基準とした各手法の単独での効果

4.2 複数の手法を組み合わせた評価

前節の結果をふまえ，各手法を組み合わせて適用する。個別の評価において結果の良くなかった (S), (M) を除き，**(1)**: (A+tol) (Qt+tol) (C+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) は (B) を，(2) は (P) を基準とした再利用適用プログラムである。RF, RB, READ/WRITE のエントリ数は，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。表 4 に示した再利用機構のオーバヘッド，およびキャッシュミス，レジスタウィンドウミスのペナルティも含めて測定を行っている。

結果を**図 4** に示す。(1), (2) のいずれの場合においても，(B) を再利用なしで実行した場合と比べて 10%以上，最大 25%程度のサイクル数削減に成功している。また，(2) では，量子化表を使用していないにもかかわらず，10〜15%程度の高速化に成功している。

なお，再利用表のエントリ数増加により結果は多少向上したが，(c) は (a) の約 20 倍の大きさを持つにもかかわらず，性能向上はわずかに 5%ほどであった。

また，(2) における命令ステップ削減数はわずかに (1) を凌駕しているが，再利用機構のオーバヘッドのため (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ヘッドが増加するためである。

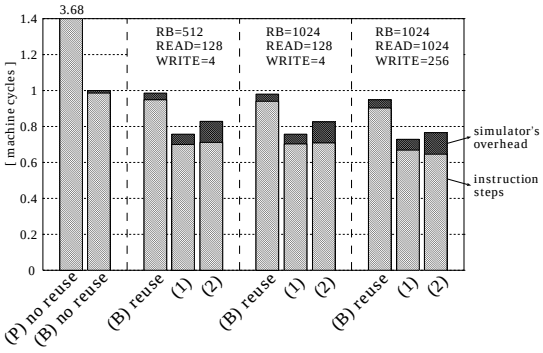

図 4 組み合わせた手法におけるサイクル数による評価

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。

5. おわりに

本稿では，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，高速化を図った。再利用可能性が高いと思われる区間を関数に切り出した上で，関数の入力値を曖昧化し，再利用率の向上による命令削減，高速化を実現した。その結果，再利用機構での実行において 10〜30%程度の命令サイクル数削減に成功した。

なお，本稿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) は，入力の局所類似性が大きいと思われるが，オーバヘッドにより再利用効果が消されている。より少ないオーバヘッドでこのような区間を切り出すことができれば，さらなる高速化が期待できる。

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, No. HPS5, pp. 1–12 (2002).
4) Álvarez, C., Corbal, J., Salamí, E. and Valero, M.: On the Potential of Tolerant Region Reuse for Multimedia Applications, *Proceedings of the 15th international conference on Supercomputing*, ACM Press, pp. 218–228 (2001).
5) 浦田敏道: 詳細 MP3 マニュアル, エム研 (1999).
6) Paul, R.: *SPARC Architecture, Assembly Language Programming and C*, Prentice-Hall (1999).