計算機アーキテクチャ 144－25
（２００１．７．２７）

MP3エンコーダの高速化

酒居 敬一[†1] 光成 滋生[†2] 成田 剛[†3]
石田 計[†4] 藤井 寛[†5] 庄司 信利[†6]

広島大学大学院 工学研究科 情報工学専攻[†1] 株式会社ピクセラ[†2]
弘前大学 理工学部 電子情報システム工学科[†3] 有限会社サムス[†4]
岡山大学大学院 自然科学研究科[†5] 株式会社コニカ[†6]

近年，マルチメディアアプリケーション向けの拡張命令をサポートした汎用プロセッサが入手できるようになってきた．PCMオーディオをmp3にする処理は遅いと感じていたし，速いエンコーダが切望されていた．そこでAMDの3D Now!やIntelのSSEという拡張命令に着目し，それらの命令をエンコーダの高速化のために使用した．

「午後のこ～だ」は，「LAME」を高速化したmp3エンコーダで，そのような命令を使うようにアセンブリ言語で書き換えたものである．さらに440BXチップセットによるIntel SMPにも着目し，マルチスレッド実行による速いmp3エンコーディングも実装してみた．

本稿では「午後のこ～だ」に実装した高速化手法や速度向上について述べる．

An Optimization of MP3 Encoder for Faster Execution

Keiichi Sakai[†1] Shigeo Mitsunari[†2] Tsuyoshi Narita[†3]
Kei Ishida[†4] Hiroshi Fujii[†5] Nobutoshi Shoji[†6]

Information Engg., Dept. of Engg., Graduate School of Hiroshima Univ.[†1] Pixela Co.[†2]
Electrical Information System Engg., Faculty of Sci. and Engg., Hirosaki Univ.[†3] Sams Inc.[†4]
Dept. of Natural Science, Graduate School of Okayama Univ.[†5] Konica Co.[†6]

Recently, some of general-purpose processors in the market have the extended instructions for multimedia applications. We feel slow to encode the PCM audio stream into the MP3, and so we want a faster MP3 encoder. We aimed at 3D Now! from AMD and SSE from Intel, and tried to use such instructions.

"Gogo-no-coda" is a MP3 encoder that is specially optimized "LAME" for speed. We rewrote the "LAME" to use such instructions in assembler by hand. We also aimed at Intel SMP with 440BX chipset. Though it seemed difficult to implement a multithreaded execution on MP3 encoding, we have done it.

In this report, we describe the optimizations used in "Gogo-no-coda" and evalue the speed-ups.

1 はじめに

製造プロセスの改良により最近の商用プロセッサはさらなる高速化と高集積化を実現し続けている．スーパースカラーによる同時命令実行数の増加やメモリアーキテクチャの改善によりアプリケーションの実行が速くなったほか，命令のスケジューリングを考慮したコード生成を行うコンパイラが一般に入手できることから充分な実行速度の得られるアプリケーションが増えてきた．それでもなお，動画像圧縮や音楽圧縮など演算量の多いアプリケーションではより多くの処理能力が必要とされていたが，このような用途ではデータの並列性が見つけやすいことから，例えばAMD 3D Now!，Intel MMX/SSEのような，SIMD的並列処理を実現する命令がプロセッサに実装されるようになった．プロセッサの改良とともに周辺チップについても，例えばIntel 440BXチップセットのようにSMP(Symmetric Multi-Processors)を安価に実現

するチップセットが発表され入手についても容易になってきた．入手性が良く安価なIntelやAMDのプロセッサを搭載したPCで実行することを仮定し，SIMD的並列処理命令を使用することによる高速化及びSMP向け高速化をMP3エンコーダに施した．

AMD 3D Now![5]やIntel SSE[7]のようなSIMD的並列処理命令がプロセッサに実装されたとしても，それらの命令を自動生成するコンパイラが入手できなかった．Intel社のコンパイラではIntrinsicにより手動で生成させることができるが，移植性や生成されるコードの速さに疑問がある．そこで，コンパイラを実装するという方向もひとつの解法であるが，本報告では人力による実装を試みた．扱うデータが16bit直線量子化PCMデータであることから，3D Now!やSSEの単精度浮動小数点演算で充分であるし，固定小数演算や整数演算命令による近似計算でも充分な場合があるので，それらの現状のコンパイラではできない高速化も行っている．440BXチップセットあるいはそれ以前のIntel SMPはバス共有型並列プロセッサであることや，MP3エンコード処理の流れから有用なマルチスレッド化エンコーディングは困難と思われていたが，エンコーダ内部のデータフローの整理などによりマルチスレッド化エンコーディングについても実装してみた．

高速化を行った「午後のこ〜だ」[3]に関して，さまざまなプロセッサにおける比較，マルチスレッド実行した場合としない場合，それぞれについての速度向上を計測した．また，必ずしもメーカ公表の最適化マニュアル[8, 6]に高速化のための情報がすべて網羅されているとは限らないため，プロセッサ内部の動作を探りながら実装したため，プログラムの改訂ごとに速くなっていることも示す．

2 MP3について

MPEG-1 Layer III(MP3)はISOにより標準化されている規格の一部(音声ストリームのみ)であり[1]，音楽等のPCMデータを非可逆圧縮することで10分の1程度に容量を減らす一方で音質の劣化が少ない特徴を持つことから広く知れ渡るようになった．一般的なMP3エンコーダのブロック図を図1に示す．本節ではHybrid Filter Bank, Psycho Acoustic Analysis, Iteration Loopおよび符号化器について順に概略を述べる．

2.1 Hybrid Filter Bank

MP3では，サブバンドフィルタとMDCTを併用するハイブリッド構成により，入力の16bit直線量子化されたPCMデータを時間領域から周波数領域に写像している．サブバンドフィルタは512タップのPFB(Polyphase Filter Bank)であり，時間領域から32の周波数帯域へ写像する．MDCTはプリエコー抑止と圧縮効率を勘案して2種類のブロック長を切替える適応ブロック長MDCTであり，長ブロック時のMDCTウィンドウ幅は36で18の周波数領域に写像されることから，PFBから出力される32の帯域全てにMDCTを施して得られる周波数領域は最終的に576となる．また，ここで使用されるMDCTは基数が36と2の巾ではないため演算に時間がかかる原因にもなっている．長短ブロック用の対称型窓関数および長短が切り替わるときの隣接ブロック間用の非対称型窓関数が用意されているが，これら窓関数用係数表を読むためのオーバーヘッドも多いため処理に時間がかかる原因でもある．MDCT出力はエイリアシング歪みを含んでいるため，MDCT出力にエイリアシング歪みを除去するバタフライ演算を施す．

2.2 Psycho Acoustic Analysis

ここでの目的はSMR(Signal Mask Ratio)を求めること，そしてプリエコー抑止のために適応ブロック長MDCT部へブロック長の切替えの指示を行うことである．およその処理は次のようになっているが，FFTや拡散関数処理やエネルギー計算のための数値演算が多くの実行時間を占めている．

1. 256点FFTと1024点FFTにより時間領域から周波数領域に写像する．
2. 各周波数帯ごとにエネルギーを求める．
3. マスキング閾値計算
4. 信号対マスク値(SMR)の計算

2.3 Iteration Loop

まず非線形量子化のため各サブバンドを3/4乗し，次の手続きでビットの割当を決定する．非線形量子化のための3/4乗演算のために多くの時間を必要としている．

1. 最大のDMR(Distortion vs. Mask Ratio)を有するサブバンドを探索する．
2. 該当サブバンドの量子化ステップを1段階小さくする．

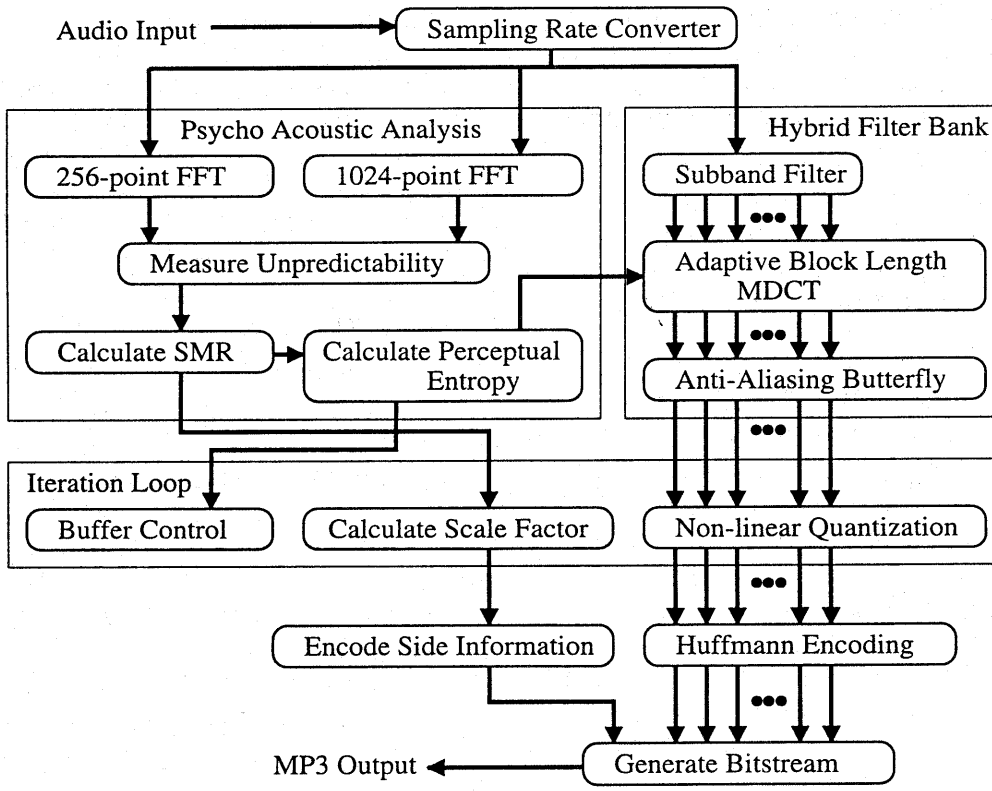

図 1: Block diagram of MP3 Encoding

3. 新しい量子化ステップに対応するSNRを選択し, 新たなDMRを計算する.
4. 現在の割当可能ビットから現在の量子化ステップに対応するビット数を減算し, 新たな割当可能ビットを計算する.

2.4 符号化器

量子化された値は, ハフマン符号化により符号化される. プロセッサのレジスタ幅よりも小さな値を多量に扱うため整数演算能力が要求される. ハフマン符号表はエンコーダとデコーダで同じ物を使用する必要があるため, 表を差し替えることによる高速化手法をとることは出来ない.

3 高速化

高速化を施す素材として, 音質が良くソースファイルが公開されているLAME[2]を選択した. LAMEはCで書かれているが現在入手可能なコンパイラではほとんど拡張命令を生成できないため, 高速化のためにはアセンブリ言語かそれに近い記述が必要である. 開発/実行環境は次のとおりで, すべて同一ターゲットアーキテクチャ(IA-32)であるが環境が異なる.

- TOWNS OS(UP)
- Windows 9x(UP)
- Windows NT(UP/SMP)
- Linux glibc-2.x(UP/SMP)
- BeOS(UP/SMP)

開発には次のような方法が考えられる.

Intel コンパイラ [10] 使用で Intrinsic 記述
Linux環境あるいはBeOS環境向けにIntelコンパイラが用意されていないため環境によっては開発ができない.

C言語ファイル中にインラインアセンブラ記述
書き方がCコンパイラに依存するため異なる環境ではコンパイルできない.

C言語とアセンブリ言語を完全に分離して記述
C言語部分はANSIに準拠した記述をすることで異なる環境でもコンパイルができる．アセンブラには，上記の複数の環境で使用できるNASM[9]がある．

これらのことから，C言語とアセンブリ言語を分離して記述することとした．NASM[9]はネットを通じてソースコードが入手可能であるため，新しいCPU向けの拡張命令の追加ができる利点も併せ持つ．

3.1 ビット処理の書き換えによる高速化

C言語ではビットフィールドがサポートされているが，その配置はコンパイラ依存であるためMP3エンコーダ内部のビット処理には不向きである．ビットごとの演算子を使用して実装すると，コンパイラにも依存するが，コンパイラにより出力される命令数が多くなり，実行時間も多くなりがちである．例えば，Huffmann符号化やビットストリーム生成のようなビット演算を多用する部分においては，プロセッサのビット演算命令を利用するようにアセンブリ言語で記述したほうが効果的である．

3.2 拡張命令使用による高速化

音楽のデータ処理を考えたとき，AMD 3D Now!やIntel SSEのような浮動小数点演算をSIMD的に行う命令が高速化には有効である．Intel MMXのような16bit固定小数点演算命令では活用できる場面が少なかったが，AMD 3D Now!命令がK6-2以降，Intel SSE命令がPentiumIII以降に実装されており，これらのSIMD的な浮動小数点演算命令を活用することで，AMD・Intel両方においてかなりの高速化ができるはずである．そこで，まずgprofによりプロファイルを獲得してこれらの命令を使用するのに有効な部分の抽出しておき，PFB, MDCT, エイリアシング歪み除去, FFTのような実行時間を多く消費する数値演算部分をアセンブリ言語により書き換えた．これらの演算はSIMD的処理が効果的であり，係数テーブルや入出力のデータの配置を変えることでload/storeの際のオーバーヘッドも削減できる．

3.3 近似による高速化

非線形量子化の際に3/4乗関数のようにプロセッサが命令でサポートしていない処理が多くの時間を消費することがある．IA-32プロセッサのIEEE準拠平方根演算はいずれも遅く，3乗して平方根を2回すると猛烈な時間を必要とする．音楽データが入力であることから厳密なIEEE浮動小数点演算は不要で，必要な精度さえ満たせば問題無いことから，拡張命令に含まれる近似演算命令を利用するか通常の整数演算命令で近似することで高速化した．特に整数演算命令による近似は高速な整数演算器を持つプロセッサでは有効と思われる[4]．

3.4 マルチスレッド実行による高速化

Intel 440BXチップセットの普及によりSMP環境を安価に入手できるようになったが，MP3エンコーディングのような時間のかかる処理をSMPを活かすようなアプリケーションは無かったため，「午後のこ〜だ」に対してマルチスレッド実装を試みた．最初はLinuxとglibc 2.0に実装されたPOSIX Thread向け，ついでWindows NT向け，後にBeOS向けなどを実装するに至った．

そこでまず対象のアーキテクチャを分析し，Intel 440BXチップセットあるいはVIA Apollo ProやServer Worksのいずれにおいても Intel P6アーキテクチャのプロセッサのSMP構成ではバス共有型であるためデータやコードの移動は最小にとどめる必要があると考えた．最小にとどめたい理由はさらに考えられ，プロセッサ内部の周波数とバスの周波数が数倍から10倍と大きくかけ離れていることの他，UMAであることからDRAMセルに起因する長いレイテンシーにプロセッサ間データ転送が律速されることへの懸念である．AMD 760MPチップセットではこのあたりの制約がずいぶんと緩和されることが予想されるが，実装の時点では入手できず本報告ではIntel系CPU向けのSMP対応にとどめた．

つぎに対象となるアプリケーション「午後のこ〜だ」を分析した．第2節でMP3エンコードの概略を述べたが，Iteration Loopにおいてbit reserverなどフレームを越えた情報のやりとりがあるため図1のブロックごとに並行して処理を行うことは不可能で，各処理の間でFIFOによりバッファしながらデータを流すことができない．フィルターバンクや聴覚心理分析においては入出力のデータ領域が大きく，プロセッサのキャッシュメモリはその大きさと比較して余裕が少なく，そういった事情からもFIFOによるデータフローは実装できそうに無いと考えた．

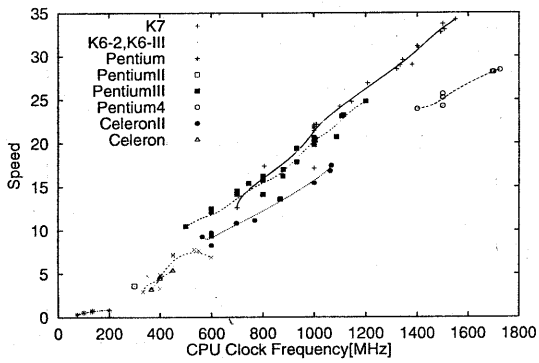

図 2: 速度向上 (聴覚心理分析あり)

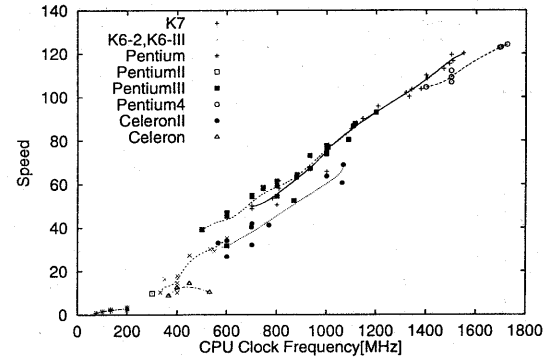

図 3: 速度向上 (聴覚心理分析なし)

これらのことから, エンコードはパイプライン処理とならざるを得ず, プロセッサ間のデータ移動を減らすためには FIFO 式パイプライン処理ではなく, 同じ処理をするスレッドが複数走行しパイプライン処理となるように mutex により処理ブロックごとに順次 lock/unlock することとした. データはフレーム単位でスレッドに与えられ, 各スレッドは mutex により他のスレッドの処理行程を追い越すこと無く全行程を処理する.

4 性能評価

MP3 エンコードに要する時間は使用する曲に依存して変化することから, 独自疑似乱数ルーチンからのデータをエンコードし結果を捨てることにした. このため再現性のあるデータが生成できることと, データを内部生成するため PCM データ入出力に関する処理時間が結果に影響せず好都合と考えた. なお, 測定は Windows 9x, Windows 2000, Linux, BeOS など, IA-32 プロセッサで動作するさまざまな OS 上で行った.

MP3 エンコードの速度向上をプロセッサファミリごとに図 2 と図 3 に示す. プロセッサの動作速度が幅広いため, 遅いプロセッサでは一定フレームの MP3 エンコード時間からの換算, 速いプロセッサでは一定時間の MP3 エンコードフレーム数からの換算で, PCM データの再生時間に対して何倍速い MP3 エンコードができるか測定している. それぞれ聴覚心理分析あり/なしのグラフで, 聴覚心理分析があるときは扱うデータ量が多いこととデータアクセスの局所性が少ない演算を含むことにより, 聴覚心理分析をしない場合に比べて遅くなっている. いずれも相違はそれほどではないがプロセッサファミリごとに拡張命令の有無やメモリアーキテクチャの改善の効果が表れている.

1 分間の PCM データの MP3 エンコードに 1 分の時間がかかった場合の速度を 1 として, CPU の種類にかかわらずクロックと実行速度のみでプロットしたものが図 5 である. クロック数の向上に従って MP3 エンコード速度が上昇しているが, この速度上昇はクロックの上昇だけではなく拡張命令の増加やキャッシュメモリの増加などプロセッサの複数の改善によるところが大きい. 高クロック時に速度上昇が鈍っているが Pentium4 や Athlon MP の登場によりいずれ改善されると考えられる.

図 4 は 1 分間の PCM データの聴覚心理分析無し MP3 エンコードに 1 分の時間がかかった場合の速度を 1 として, Dual PentiumIII 構成の PC でシングルスレッド実行とマルチスレッド実行を比較したものである. PentiumIII による SMP は, バス共有型であることからプロセッサ間のデータ移動にはプロセッサ内部のデータパスと比較して猛烈に時間がかかる. 「午後のこ〜だ」ではデータパスの工夫により内部のデータ移動を少なくした.

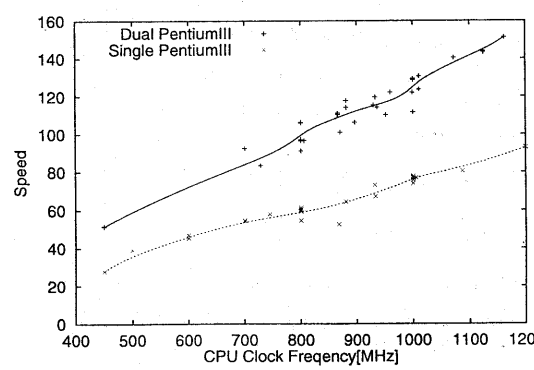

図 4: Single/Dual プロセッサの速度比較

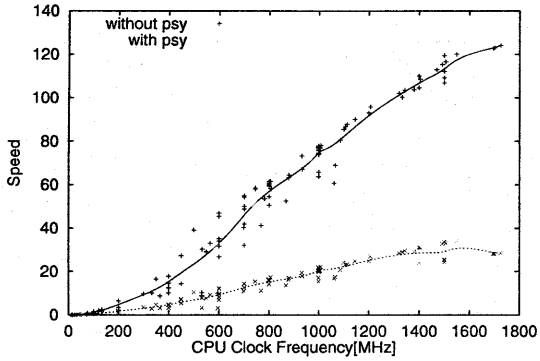

図 5: クロック周波数に対する速度向上

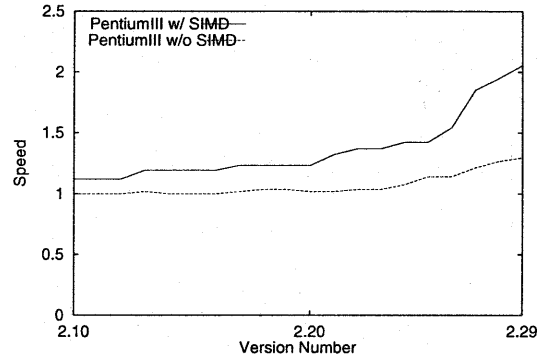

図 6: バージョン改訂に伴う速度向上

さらに入力データのようなキャッシュすることが無効かむしろ有害でさえある場合に「キャッシュしない」ことを明示する命令も使用した．このためクロック数の増加に伴う速度増加が得られるようになったが，それでもなお高クロック領域で伸び悩んでいる．この原因のひとつに現行のバス共有型方式があると思われる．実際 (報告例はまだ少ないが) 異なる方式を用いた Athlon MP や Pentium 4 などの SMP 構成では大幅な上昇が見られる．

プロセッサの製造メーカは最適化マニュアルを公開しているが，競合メーカとの戦略上すべてを明かしてはいない．そのため，高速化の過程では推測を伴うようなコーディングが要求されることがある．図 6 は，「午後のこ〜だ」を最初に公開した Ver2.10 からおおよその高速化が終わった Ver.2.29 までの MP3 エンコード速度の推移である．ここでのバージョン推移はアルゴリズムの高速化のみを念頭におき，生成データの音質的劣化を起こすような演算処理の省略は行っていない．Ver.2.10 で PentiumIII 特有の SIMD 処理を使わなかった場合を 1 としている．Ver.2.29 の後，若干のバグフィックスと機能追加を行い，現時点の最新版は Ver.2.39 である．本来，機械的にコード生成は行われるべきであり，そうであるがゆえにコンパイラを作成することが必須であったが，必ずしも必要な情報が公開されていないことから人力による試行錯誤の高速化も意味があったと考えている．

5 おわりに

IA-32 をターゲットに MP3 エンコーダ LAME を高速化し「午後のこ〜だ」として広く使われるようになった．「午後のこ〜だ」では VBR(Variable Bit Rate) エンコーディングもサポートしているが，最近の LAME と比較して処理速度や音質に問題がある．今後は，最近の LAME を Pentium4 や Athlon MP 向けにも高速化する．

謝辞

GPL に基づき MP3 エンコーダ LAME を開発している方々，「午後べんち」への参加者，NASM を開発している方々へ深く感謝致します．

参考文献

[1] ISO/IEC 11172-3, “Coding of Moving Pictures and Associated Audio for Digital Strage Media at up to about 1.5Mb/s - Part 3: Audio”, Aug, 1993.

[2] “The LAME Project”, http://www.mp3dev.org/mp3/

[3] “午後のこ〜だ”, http://homepage1.nifty.com/herumi/soft.html

[4] Keiichi Sakai, Itaru Fujiwara and Tadashi Ae: “Extended VLIW Processor for Real-Time Imaging”, Real-Time Imaging V, Proceedings of SPIE, Vol.4303, pp43-50 (2001).

[5] *AMD Extensions to the 3D Now! and MMX Instruction Sets Manual*, AMD, 2000.

[6] *AMD Athlon Processor x86 Code Optimization Guide*, AMD, 2000.

[7] *The Intel Architecture Software Developers Manual, Volume 1-3*, Intel, 1999.

[8] *Intel Architecture Optimization Reference Manual*, Intel, 1999.

[9] “Netwide Assembler”, version 0.98, http://www.web-sites.co.uk/nasm/, June 1999.

[10] “Intel Compiler”, http://developer.intel.com/software/products/compilers/.