MP3エンコーダの高速化実装

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]

近年の汎用パーソナルコンピュータに多く使われているIA-32プロセッサは過去との互換性のために命令体系はCISC的である．しかし内部動作はRISCマイクロ命令への変換，ハイパーパイプライン，アウトオブオーダ，などRISC的アーキテクチャが多数取り入れられている．さらにSIMD的演算命令の搭載によりデータ並列処理を実現している．とはいえその新しい機構に応じたコードを生成するコンパイラはまだまだ少数であり，また対応していたとしてもコンパイラ独自の拡張C言語による記述が必要であることが多い．そのため通常のC言語による記述を主体としたベンチマークではプロセッサの正当な評価を行いにくい．そこで我々は実用的なアプリケーションとしてMP3エンコーダを選択しコード全般にわたってアセンブリ言語による最適化処理を行った．その結果C言語によるコードに対し2倍から3倍の高速化を達成した．

A Performance Centric Optimization of MP3 Encoder Programs

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

The architecture of IA-32 processors, which are recently used in personal computers in general, can execute the industry-standard x86 instruction set like CISC for binary compatibility. But in fact, the processors process simpler RISC micro operations instead of complex x86 instructions, and support hyper pipelined technology and out-of-oder speculative execution. IA-32 processor also has the SIMD instructions. However, there are few compilers which generate the code supporting the new technology, or we must use the extended C language, if any. Therefore it is difficult to evaluate the processor exactly by the benchmark based on the ordinary C language. Then we chose a MP3 encoder for practical application and rewrote the main routines of the encoder by assembly language. As a result, we have achieved about two or three times faster optimization.

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十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，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，たとえばAMD 3D Now!™命令，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，たとえばIntel®440BXチップセットのようにPC向けに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およびSMP向け高速化をMP3エンコーダに

†1 広島大学大学院工学研究科
Department of Engineering, Graduate School of Hiroshima University
†2 株式会社ピクセラ
Pixela Co.
†3 株式会社インテリジェントシステムズ研究開発部
INTELLIGENT SYSTEMS Co., Ltd. R&D
†4 有限会社サムス
Sams Inc.
†5 岡山大学大学院自然科学研究科
Department of Natural Science and Technology, Graduate School of Okayama University
†6 株式会社コニカ
Konica Co.

施した．

ただし，AMD 3D Now!™ [8] や Intel®SSE® [10] 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存在しなかった．そこでコンパイラを実装するという方向があるが，そのためにも生成されるコードの比較対象が必要であり，コード生成のためにプロセッサメーカの公表する資料[9),11)] に記述されていないことを推測するためにも，人手による実装をまず試みることにした．

対象とする IA-32 互換プロセッサどうしの比較のために SIMD 的並列処理命令を使う場合だけではなく，使わない場合についても実装した．扱うデータが 16 ビット直線量子化 PCM データであることから，AMD 3D Now!™ 命令や SSE®命令の単精度浮動小数点演算で十分であるし，固定小数演算や整数演算命令による近似計算でも十分な場合があるので，現在入手できる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，キャッシュ制御命令などの評価のために実装を試みた．

そのようにして実装した MP3 エンコーダ「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，それぞれについて速度向上を評価した[14]．

2. 背　　景

汎用プロセッサに採用されている高速化技術と実験対象である MP3 エンコーダについて述べる．

2.1 高速化技術

デザインルールの改良とともにプロセッサは高集積化と高クロック化の方向へ進んでいる．高集積化により多くの演算器やレジスタを搭載することができ，それらを高クロックでパイプライン動作させることで高いスループットが得られるようになっている．汎用スーパースカラプロセッサでは，コードを並列処理するため命令の Out-of-Order 実行を行い，データの並列処理のために同時に複数のデータに対して同じ演算を行える命令を装備している．

最近の汎用プロセッサに見られるような段数の多いパイプライン動作では，条件分岐命令実行時やメモリ操作命令実行時に発生するパイプラインの乱れが非常に大きいという問題がある．そのような問題についても，(静的および動的) 分岐予測と投機的実行により演算パイプラインの乱れを最小限に抑え，長いレイテンシによる処理速度の低下を減らすように実装されている．さらに，レイテンシやスループットの異なる命令を混在させて効率良くパイプライン実行することは困難になってくるため，命令ストリームからの命令はパイプラインへ投入しやすくなるように，より細かい命令に分解されている場合がある．

近年の PC に多く使われている Intel®Architecture 32（IA-32）プロセッサは，過去との互換性のために命令体系は CISC 的である．しかし内部動作は RISC マイクロ命令への変換，スーパースカラ，スーパーパイプライン，など RISC 的アーキテクチャが多数取り入れられている．さらに SIMD 的演算命令の搭載によりデータ並列処理を実現している．

PC 向けのチップセットにはバス共有型であるが SMP 構成可能なものが出現している．たとえば，Intel®440BX チップセットではプロセッサ間のデータ移動には主記憶を使う必要があり 100 MHz の SDRAM により帯域幅が 800 MB/秒に制限されるが，使い方によっては高速化できる可能性がある．

2.2 命令セット

IA-32 命令セットのうち，整数演算器（IU）で処理される命令（IU 命令）や浮動小数点数演算器（FPU）で処理される命令（FPU 命令）は 80386 世代から存在するが，Pentium®世代後期から Intel®MMX® テクノロジ命令（MMX 命令），Pentium III 世代から Intel®ストリーミング SIMD 拡張命令（SSE 命令），Pentium 4®世代から Intel®SSE2 命令が追加されている．AMD 社からの IA-32 互換プロセッサでは，それらに追従しながらも 3D Now!™ テクノロジ命令（3D Now! 命令）や Enhanced 3D Now! 命令のような独自の拡張命令を実装してきた．これらのプロセッサでは，cpuid 命令が使用可能であれば cpuid 命令で拡張命令セットが実装されているかどうか知ることができる．cpuid 命令が使用できないプロセッサの場合はこのような拡張命令セットは実装されてない．

MMX 命令では 80 ビット幅の FPU レジスタを複数の整数を格納する 64 ビットレジスタとして使用し（mm0〜mm7），8 ビット × 8 並列，16 ビット × 4 並列，32 ビット × 2 並列，64 ビットのいずれかのオペランドサイズを命令語で指定ながら使用する．

3D Now! 命令では 80 ビット幅の FPU レジスタを 2 つの 32 ビット単精度浮動小数点数を格納するレジスタと見なして，2 並列で演算する命令の拡張である．3D Now! 命令と MMX 命令はいずれもほとんどの命令は第 1 オペランドにはレジスタのみ，第 2 オペラン

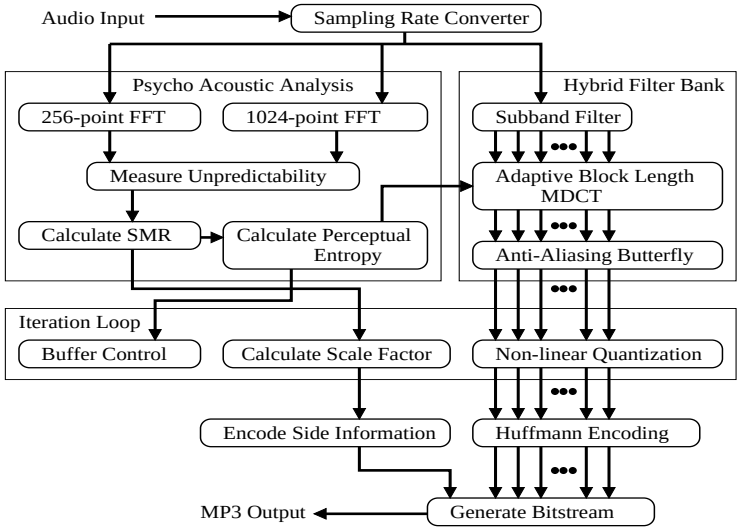

図 1 MP3 エンコードのブロック図
Fig. 1 Block diagram of MP3 encoding.

ドにはレジスタ/メモリを指定可能である．Athlon^{TM} や K6-III+^{TM} などのプロセッサでは Enhanced 3D Now! 命令として 5 命令が追加されている．

MMX 命令と 3D Now! 命令いずれも FPU レジスタを共用するため切替えを明示する命令 [emms, femms] が必要であるが，プロセッサが MMX 命令や 3D Now! 命令を実行可能であれば OS のサポートなしにアプリケーションで使用可能である．MMX レジスタと 3D Now! レジスタはスタック形式ではないのでプログラムが組みやすい．

SSE 命令は単精度浮動小数点数を 4 並列で演算する拡張命令セットである．SSE 命令セットから見たレジスタ（xmm0〜xmm7）は MMX や 3D Now! のように FPU レジスタを利用しているのではなくまったく新規に作られた 128 ビットレジスタを使用する．そのため MMX 命令や 3D Now! 命令とは異なり [emms, femms] が不要であるがコンテキストスイッチなどの際に OS が SSE レジスタをサポートする必要があり，SSE 命令が実装されたプロセッサであっても必ずしも利用できるとは限らない．SSE 命令に関しては，MMX 命令や 3D Now! 命令と異なり，cpuid 命令で得られる情報によりプロセッサに実装されてることが分かっても OS が SSE 命令を使用可能にしていない場合がある．SSE 命令は基本的に SSE レジスタの最下位データのみを演算対象とする命令には ss，4 つの（パックされた）データを演算対象とする命令には ps がつき，4 並列演算のほかに命令形式の異なる FPU のように使用することができる．たいていの SSE 命令は ss より ps の方がレイテンシは 1[clocks per instruction] 増加，スループットは 1[instruction every clock] から 1/2[instruction every clock] 低下する．

2.3 MP3 エンコーダについて

MPEG-1 Layer III（MP3）は ISO により標準化されている規格の一部（ここでは音声ストリームのみ）であり[1)]，音楽などの PCM データを非可逆圧縮することで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わたるようになった．エンコーダは数種類のサンプリング周波数を受付けるが，音楽用 CD では音楽はサンプリング周波数 44.1 kHz の 16 ビット直線量子化ステレオ PCM により符号化されているため MP3 エンコーダの入力にも同様の PCM データがよく使用される．サンプリング周波数 44.1 kHz の 16 ビット直線量子化ステレオ PCM データ入力に対しては音質と出力ビットレートの兼ね合いから 128 k[bps]（[bit per second]）の出力ビットレートで使用されることが多い（圧縮率は 128 k/(44.1 k*16*2) ≒ 0.09）．そこで本論文では特に断わらない限りはサンプリング周波数 44.1 kHz の 16 ビット直線量子化ステレオ PCM データを入力，MP3 で 128 k[bps] 出力の場合について述べる．

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．

2.3.1 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 16 ビット直線量子化 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，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り替える適応ブロック長 MDCT で，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 の帯域すべてに MDCT を施して得られる周波数領域は最終的に 576 となる．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．

PFB と MDCT ともに計算量が多いこと，特に長ブロック用 MDCT は基数が 36 であり 2 のべき乗ではなく効率良く処理できないこと，長短ブロック用の対称型窓関数および長短ブロック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を必要に応じて読むためのオーバヘッドが多いこと，により汎用プロセッサによる処理には時間がかかる．

2.3.2 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へ写像する．
(2) 各周波数帯ごとのエネルギーを計算する．
(3) マスキング閾値を計算する．
(4) 信号対マスク値（SMR）を計算する．

2.3.3 Iteration Loop

まず非線形量子化のため各サブバンドの各要素を 3/4 乗し，次の手続きでビットの割当てを決定する．非線形量子化のための 3/4 乗演算，ビット割当てに処理時間を要する．

(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．
(3) 新しい量子化ステップに対応する SMR を選択し，新たな DMR を計算する．
(4) 現在の割当て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，新たな割当て可能ビットを計算する．

2.3.4 符号化器

量子化された値は，ハフマン符号化される．その際に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に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もの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できない．

3. 実　　験

MP3 エンコードは IA-32 の FPU 命令や IU 命令だけではかなりの時間がかかってしまう処理であり，3D Now! や SSE 命令（以下，拡張命令）を使うことで高速化が望めそうな対象でもあることから，ソースファイルが公開されている MP3 エンコーダ LAME [2] を対象に実験を行った．本論文ではコードサイズの増減をいっさい無視し速度向上のみに着目した最適化を「高速化」と記述することにする．たとえば，通常のコンパイラでは一定の上限を設けてコード量の増加を抑えているが，ループ展開やインライン展開のようなコード量増加をともなう最適化であっても，速度が向上する限り展開を行う．

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）

SSE および SSE2 拡張命令に対応している Intel コンパイラ [13] が上記すべての環境において使用可能であるとは限らないこと☆，上記すべての環境において同じ C コンパイラを使用できるとは限らないこと，アセンブラは上記すべての環境において使用可能な NASM [12] が存在すること，により 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．C 言語記述は原則として ANSI 標準に準拠するが，ANSI 非準拠のコンパイラ固有機能を使用することで速度向上が望める場合は，条件コンパイルで対応することとした．NASM は Intel 標準のオペランド順で記述可能でプロセッサの命令表との整合が良く，ネットを通じてソー

☆ 現在では Windows 版のほかに Linux 版も存在する．

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，という利点がある．実際に我々は Enhanced 3D Now! 命令と SSE2 命令に対応するように改造して使用している．

時間を消費する部分の抽出は gprof のようなプロファイラを使用するとともに，関数個別の高速化の場合にはプロセッサが持つ Time-Stamp Counter を RDTSC 命令で関数の実行前後に読み出して所要クロック数を測定している．Time-Stamp Counter を用いると，命令の配置最適化や，変数や定数データのアクセス最適化などの最適化にも役立つ．

通常のコンパイラで行っている最適化だけではなく[7]，関数やソースファイルを越えたコード移動やデータ移動，データ構造の変更，変数のアライメント調整，演算精度の変更，を行うとともに拡張命令への置換えを行っている．拡張命令を実行できないプロセッサでも拡張命令を使用しないアセンブリ言語記述関数が C 言語記述関数よりも速く実行できる場合が多いため，拡張命令を使用しないアセンブリ言語関数も用意することにした．高速化については次に述べる．

例として C 言語で記述された非線形演算ルーチンを次に示し，説明する．

```
void calc_pow075_C(float *xr, float *xrpow)
{
    int     i;
    float   tmp;
    for( i = 0; i < 576; i++ ){
        tmp = fabs( *xr++ );
        *xrpow++ = sqrt( sqrt(tmp) * tmp );
    }
}
```

C 言語ルーチンを FPU により演算するようにアセンブリ言語で書くと次のようになる．コンパイラが生成すると命令順に関して最適に配置するとともに loop unrolling するが，fsqrt 命令の実行時間が支配的であるため，ここでは説明のためにあえて単純なアセンブリ言語記述を示す．パイプライン命令である乗算命令 fmul（レイテンシ 5，スループット 1/2）や加算命令 fadd（レイテンシ 5，スループット 1/1）と比較して，fsqrt 命令は非パイプラインかつ長レイテンシ（約 20）な命令なので，実際には使用していない．

```
calc_pow075_FPU:
    mov     eax,[esp+4] ;eax=xr
    mov     edx,[esp+8] ;edx=xrpow
    mov     ecx,576
    jmp     short .lp0
    align   16  ; ループ開始を 16 バイト境界に整合
.lp0:
    fld     dword [eax] ;TOS(Top Of Stack)=*xr
    add     eax,4       ;xr++
    fabs            ;TOS に対して絶対値演算
    fld     st0     ;TOS(=*xr) を push する．
    fsqrt           ;TOS に平方根演算．
    fmulp   st1,st0;*xr と TOS を乗算し 3/2 乗を得る
    fsqrt       ;TOS を平方根演算し 3/4 乗を得る
    fstp    dword [edx] ;TOS を*xrpow に格納する．
    add     edx,4   ;xrpow++
    dec     ecx
    jnz     .lp0
    ret
```

3.1 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，浮動小数点演算を SIMD 的に行う拡張命令が高速化には有効である．MMX 命令のような 16 ビット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，3D Now! 命令が K6-2 以降，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．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 の際のオーバヘッドも削減できる．

例に示された C 言語プログラムを SSE を使用するようにアセンブリ言語で書き換えた例を次に示す．逆数平方根演算命令を使用するため入力値 xr[] が 0 に近いとき誤差が大きいと思われたが，実用上は問題なかった．逆数平方根演算命令 rsqrtps はレイテンシ 2，スループット 1/2 であるため fsqrt 命令よりも速い．

```
calc_pow075_SSE:
    movlps  xmm7,[D_ABS] ; 符号ビットの零クリア用
    mov     eax,[esp+4] ;eax=xr
    mov     edx,[esp+8] ;edx=xrpow
    mov     ecx,576/8 ;loop unrolling で 8 要素/周
    movlhps xmm7,xmm7
    jmp     short .lp0
    align   16  ; ループ開始を 16 バイトに整合
.lp0:
    movaps  xmm0,[eax] ;xr から順に xmm0,xmm1 へ
    movaps  xmm1,[eax+16]
    add     eax,32    ;xr を 8 要素分進める
    andps   xmm0,xmm7 ;abs(), 符号ビットを 0 にする
    andps   xmm1,xmm7
    rsqrtps xmm2,xmm0 ; 逆数平方根命令により
    rsqrtps xmm3,xmm1     ;x から 1/√x を得る
    mulps   xmm0,xmm2 ; 乗算命令により
    mulps   xmm1,xmm3     ;x × 1/√x で √x を得る
    rsqrtps xmm2,xmm2 ; 逆数平方根命令により
    rsqrtps xmm3,xmm3     ;1/√x から √√x を得る
    mulps   xmm2,xmm0 ; √x × √√x から
    mulps   xmm3,xmm1     ;x の 3/4 乗を得る
    movaps  xmm0,xmm2
    movaps  xmm1,xmm3
```

```
    cmpps   xmm0,xmm0,0 ; 逆数平方根命令で
    cmpps   xmm1,xmm1,0     ;NaN になった？
    andps   xmm2,xmm0 ; もし演算途中で NaN なら
    andps   xmm3,xmm1     ; 結果を零にする
    movaps  [edx],xmm2 ;xmm2,xmm3 から順に xrpow へ
    movaps  [edx+16],xmm3
    add     edx,32  ;xrpow を 8 要素分進める
    dec     ecx
    jnz     .lp0
    ret
```

3.2 近似による高速化

非線形量子化の際に 0.75 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．SSE あるいは 3D Now! が使用できないときは IU 命令か FPU 命令を使用するしかなく，通常は浮動小数点演算器があれば浮動小数点演算は浮動小数点演算器に処理させるほうが速いのであるが，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 回すると遅くなる．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ないことから，拡張命令に含まれる近似演算命令を利用するか通常の IU 命令で近似することで高速化した．特に IU 命令による近似は高速な IU を持つプロセッサでは有効と思われる[6]．

浮動小数点数を 0.75 乗演算しやすいように指数部を 4 の倍数となるように仮数部を調整しそれぞれ分けたあとで指数部は 0.75 倍，仮数部は近似により 0.75 乗している．$y = x^{0.75}$ を求めるとき，$0 < x < 2$ である x に対して $b = x - 1$ とすると 2 項定理により級数展開を得ることができる．単精度浮動小数点数は符号 1，指数 8，仮数 23 の計 24 ビットであるが，仮数部の最上位は 1 となるように正規化されているため，仮数部は 24 ビット分の精度がある．速く演算するために指数部と仮数部の分離後の仮数部の範囲を 3 つに分割して近似するとともに，非線形演算の出力精度は前後の行程を考えて 16 ビット程度で十分なので第 4 項までの処理で演算を打ち切った．第 4 項までの近似式を次に示す．

$y = 1 + b*3/(4) - b^2*3/(4^2*2) + b^3*3*5/(4^3*2*3)$

仮数部の演算は IU 命令による固定小数点演算である．実装例を次に示す．

```
; 指数は単に 0.75 することにして，端数は仮数に乗算する．

; このとき-0.125 <= b <= 0.125 に収めるために次の 3 区間に分離．
;     1.0 <= 仮数 <= 1.28125(1.01001b)
;         補正のための係数 = 0.875(14/16)
;     1.272727 < 仮数 <= 1.625(1.101b)
;         補正のための係数 = 0.6875(11/16)
;     1.555555 < 仮数 < 2.0
;         補正のための係数 = 0.5625(9/16)
; これら係数は pow075_table0 と pow075_table1 に置いている
        align 16
calc_pow075_NONE:
    push    ebp
    push    ebx
    push    esi
    push    edi
    mov     ebx,576
    jmp     short .lp0
    align 16
.lp0:   ; ecx = x = *xr++
    mov     esi,[esp+4*4+4]
    mov     ecx,[esi+ebx*4-4]
    xor     edx,edx
    rol     ecx,9   ;cl に指数部を取り出す
    test    cl,cl
    jz      near .store ; 非正規化数なら終了
    cmp     cl,255
    je      near .store ;NaN なら終了
    mov     edi,ecx
    stc
    rcr     edi,1   ; 仮数部の最上位の 1 を補う
    shr     edi,8   ; 仮数，9.23 形式
    xor     ebp,ebp ; 近似区間を分割，ebp=区間
    cmp     edi,0x00A40000  ;1.28125，9.23 形式
    adc     ebp,edx ; 比較結果の CF により区間決定
    cmp     edi,0x00D00000  ;1.625，9.23 形式
    adc     ebp,edx ; 比較結果の CF により区間決定
    movzx   eax,byte [pow075_table1 + ebp]
    shl     ebp,4
    mul     edi     ; edx:eax = 37.27 形式
    lea     edi,[eax-(1<<27)]
; 仮数部の近似演算開始
; y = 1
    mov     esi,(1<<29) ; esi = 1.0，3.29 形式
;   + b*3/(4)
    mov     eax,edi
    shl     eax,1
    add     eax,edi ; eax = 3.29 形式
    add     esi,eax ; esi = 3.29 形式
;   - b*b*3/(4*4*2)
    imul    edi     ; edx:eax は 5.59 形式
    shl     edx,2
    sub     esi,edx
; 第 3 項↑までで誤差 1 万分の 1 以下
;   + b*b*b*3*5/(4*4*4*2*3)
    mov     eax,0x6AAAAAAA  ; 5/(4*3)，0.32 形式
    imul    edx     ; edx:eax は 3.61 形式
    mov     eax,edx
    imul    edi     ; edx:eax は 8.56 形式
    shl     edx,5
    add     esi,edx
; 第 4 項↑までで誤差 10 万分の 1 以下
; ここで esi は仮数 (3.29 形式)
; cl に指数 (offset binary)
    sub     cl,127  ; 指数を計算
    xor     eax,eax
    mov     al,cl
    and     al,3
```

```
    sar     cl,2
    mov     eax,[pow075_table0+ebp+eax*4]; 補正
    imul    esi ;1.0 <= edx:eax < 8.0, 9.55 形式
; 指数と仮数を正規化して IEEE754 形式に戻す
    xor     eax,eax ; 符号ビットは正 (=0)
    mov     al,cl
    add     cl,cl
    add     al,cl   ;al=正規化前の指数 (2's comp.)
    bsr     ecx,edx
    sub     cl,23
    add     al,cl
    add     al,127  ; 指数の正規化 (offset bin.)
    shr     edx,cl  ; 仮数の正規化
    and     edx,0x007fffff
    shl     eax,23
    or      edx,eax ;or して指数と仮数を混合
.store:
    mov     edi,[esp+4*4+8] ; *xrpow++ = edx
    dec     ebx
    jnz     near .lp0
.return:
    pop     edi
    pop     esi
    pop     ebx
    pop     ebp
    ret
```

3.3 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．たとえば，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．

比較結果によって整数変数を 0 や 1 にセットしたり+1 するような場合，比較命令による比較の結果がキャリーがセット/リセットされるような形に変換できれば，条件付き命令が使用不可能なプロセッサでも，条件分岐なしにキャリーを使用する除算命令かビット操作命令により汎用レジスタに持ってくることができる．キャリーを利用して条件分岐をなくした例は，さきほどの近似による書換え例のうち，近似区間分けのところで使用している．このような書換えは条件分岐命令によるペナルティを減らすだけではなく，Branch Target Buffer を消費しない利点がある．

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パーソナルコンピュータとして SMP 環境を入手できるようになった．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なかったため,「午後のこ〜だ」に対してマルチスレッド実装を試みた．440BX による SMP はバス共有型であるためプロセッサの動作速度（たとえば 800 MHz）と主記憶の速度（100 MHz）の差が大きく，プロセッサ間のデータ移動はプロセッサ上のキャッシュからのデータ移動に比べて遅い．単一プロセッサで動作するアプリケーションを単純にジョブ分割するとプロセッサ間データ移動が発生するとそれだけ処理時間が増えるので，キャッシュ制御命令によるバス帯域やキャッシュ容量の効率的利用，必要バス帯域を減らすジョブ分割を行った．まず Linux と glibc 2.0 に実装された POSIX Thread 向け，次いで Windows NT 向け，後に BeOS 向けなどを実装するに至っている．

MP3 エンコードの概略はすでに述べたが，Iteration Loop における bit reserver のようにフレームを越えた情報のやりとりがあるため図 1 のブロックごとにスレッドを割り当てて並行して処理を行うことができない．そこでデータはフレーム単位でスレッドに与え各スレッドは mutex により他のスレッドの処理行程を追い越すことなく全行程を処理することにし，Linux カーネルではできるだけスレッドが CPU 間を移動しないようスケジュールされるためプロセッサ数とスレッド数を同じにして生成することとした．プロセッサ数=スレッド数=2 の場合を**図 2** に示し，データフロー（図中の矢印）を次に説明する．

サンプリング周波数 44.1 kHz の 16 ビット直線量子化ステレオ PCM データを入力とする場合，左右の入力各 1,152 サンプルに対して MP3 フレームを 1 つ形成する．576 サンプルを 1 グラニュルと呼ぶ．サンプリング周波数 44.1 kHz の 16 ビット直線量子化ステレオ PCM データ入力に追従しているときは毎秒約 38 フレームが出力されることになる．1 フレームを形成するとき PFB 開始位置や MDCT 開始位置や FFT 開始位置は**図 3** のようにそれぞれ遅延を持っている．long FFT の場合は 1,024 サンプルを入力とするが，図 3 のように PFB 入力開始位置（=出力位置）とは大きくずれているため，入力 PCM データに関してシフトレジスタ的なバッファを持っている．MDCT 入力は 1 グラニュル前の PFB 出力も必要とするため，PFB 出力に関しても 1 グラニュル分を余分に保持するためのバッファを持つ．遅延はデコーダに合わせるため固定であり，入力 PCM データ用バッファは左右それぞれ 1,904 サンプルで 1 サンプルあたり 16 ビットなので 7,616 バイト，PFB 出力バッファは左右それぞれ 1,728 サンプルで単精度浮動小数点数で 1 サン

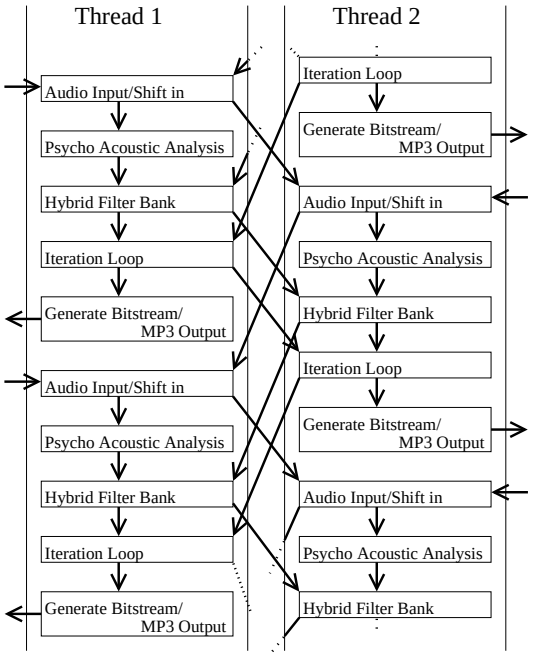

図 2 スレッド構成
Fig. 2 Thread construction.

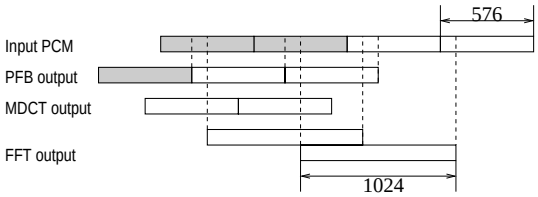

図 3 バッファ構成
Fig. 3 Buffer construction.

プルあたり 4 バイト必要なので 13,824 バイトである．

図 3 で灰色で示される部分が前フレームからの持ち越しであり，入力 PCM データ用バッファのうち 3,008 バイトと MDCT への入力のうち 4,608 バイトがスレッド間でデータ移動が必要でキャッシュに入ってない．あわせて 12,224 バイトが主記憶から読まれることとなる．

もし，フレームを越えた処理を適切にバッファしてブロックごとに分割した MIMD 並列が実装できたとすると，単純にスレッド間のデータ移動のために PCM データ入力（4,608 バイト）と PFB 入力（9,216 バイト）と MDCT 入力（13,824 バイト）と非線形量子化（9,216 バイト）と FFT 入力（12,800 バイト）の合計 49,664 バイトの主記憶のアクセスが発生することから，データ移動に関しては図 2 のほうが良いことになる．

主記憶経由でプロセッサ間データ移動が必要なデータ要素に関してはキャッシュ制御命令を使用することにしてキャッシュの利用効率を上げることにする．キャッシュ制御の必要な部分はアセンブリ言語により記述している．

4. 実験結果

まず MP3 エンコードの速さを測定し，次に実行時間を占める関数について個別に速度向上を数例測定した．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[4]．これにより再現性のあるデータで測定でき，PCM データ入力に関する処理時間が結果に影響しない．

MP3 エンコードの速さの測定は IA-32 プロセッサで動作するさまざまな OS 上で，多くのユーザの手により行われた[4]．時間は MP3 エンコーダ内部でエンコード開始時刻と終了時刻から計算しているが，プロセッサの演算能力が幅広いことと OS の時計の精度が良くないため，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をもって速さとしている．たとえば，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は速さが 1 である．

MP3 エンコードの速さをプロセッサファミリごとに**図 4** と**図 5** に示す．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演算量が多いことにより，聴覚心理分析をしない場合に比べて遅くなっている．いずれも相違は大きく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．

CPU の種類にかかわらずクロックと速さのみでプロットしたものが**図 6** である．クロック数の向上に従って MP3 エンコードが速くなっているが，これはクロックの上昇だけではなく拡張命令の追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．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 7 は，PentiumIII の Dual 構成 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をまたぐデータ参照が発生しシングルプロセッサ機と比較して時間がかかる．「午後のこ〜だ」ではデータパス周りの工夫により共有メモリの

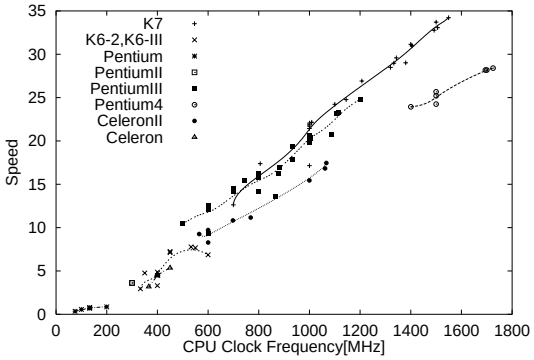

図 4 MP3 エンコード速さ（聴覚心理分析あり）
Fig. 4 Encoding speed (with psycho achoustic analysis).

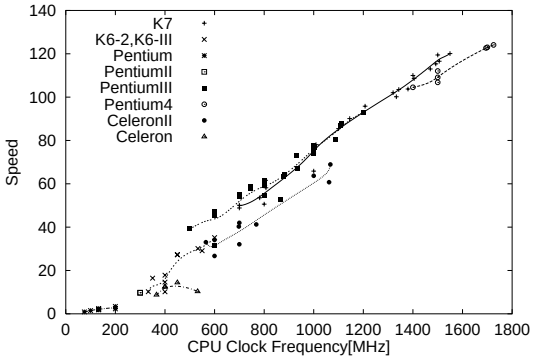

図 5 MP3 エンコード速さ（聴覚心理分析なし）
Fig. 5 Encoding speed (without psycho achoustic analysis).

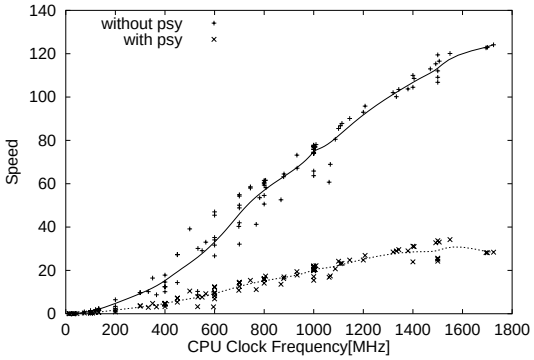

図 6 クロック周波数に対する速さの向上
Fig. 6 Speed-up vs. clock frequency.

データ移動を少なくするとともに，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．なおキャッシュへ先行して読み込むキャッシュ制御命令もあるが，入力 PCM データなどはキャッシュしても再利用されないためキャッシュ容量を消費するだけで効果がなかったので使用しなかった．

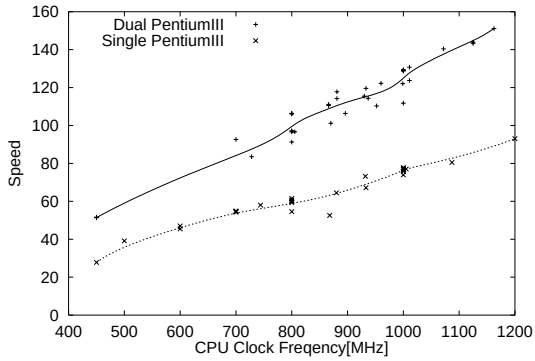

図 7 Single/Dual プロセッサの速さ比較
Fig. 7 Comparison of encoding speed by single processor with dual processor.

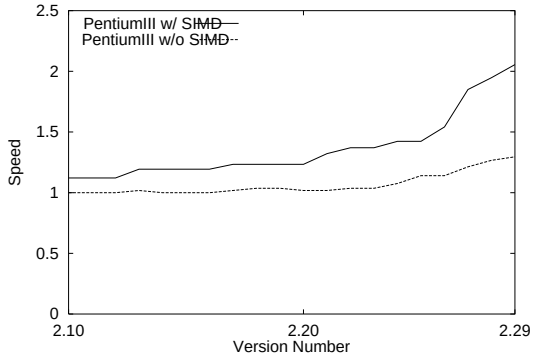

図 8 バージョン改訂にともなう速さ向上
Fig. 8 Speed-up vs. revision.

実験の結果では 3D Now! や SSE 命令はアプリケーションの実行時間短縮にはおおいに役立っている（**図 8**）．しかし，アセンブリ言語による書き換えを行った関数の実行時間短縮には関数ごとに大きな相違が見られた．実行時間の大半を占める 4 関数について，2 種類のプロセッサにおける所要クロック数を測定した結果を**表 1** と**表 2** にそれぞれ示す．IU/FPU は IU および FPU を使用する関数による MP3 フレーム 1 個生成時の当該処理に要するクロック数，SSE および 3D Now! はそれぞれのプロセッサの拡張命令を使用した関数により MP3 フレーム 1 個生成時の当該処理に要するクロック数，速度向上はこれらの比である．測定は 44.1 kHz ステレオ 16 ビット直線量子化 PCM データ 10 分間相当分をエンコードして MP3 フレームを 22968 生成する処理を 10 回繰り返したときの平均クロック数である．

表 1 の中の 0.75 乗演算は FPU 命令を使って処理すると 2,408 クロックと遅かったため IU による結果

表 1 関数ごとの実行時間（Intel PentiumIII 700 MHz）
Table 1 Elapsed clocks for each function (Intel PentiumIII 700 MHz).

	IU/FPU $[\times 10^3 clk]$	SSE $[\times 10^3 clk]$	Speed-up
1 フレーム	20834	13191	1.58
PFB	2287	850	2.69
1024-FFT	2098	1055	1.99
MDCT	1276	502	2.54
0.75 乗演算	1280	88	14.48

表 2 関数ごとの実行時間（AMD Athlon 700 MHz）
Table 2 Elapsed clocks for each function (AMD Athlon 700 MHz).

	IU/FPU $[\times 10^3 clk]$	3D Now! $[\times 10^3 clk]$	Speed-up
1 フレーム	19890	13305	1.50
PFB	1918	725	2.65
1024-FFT	1734	1018	1.70
MDCT	1027	566	1.81
0.75 乗演算	1197	354	3.38

をのせている．3 章で例にあげた 0.75 乗演算は Intel PentiumIII プロセッサ向けであるが，表の他の関数に比べて速度向上が大きい．これは，FPU で実行するときの fsqrt 命令が，SSE 命令に書き替えたあとの rsqrt 命令に比べて約 20 倍のレイテンシを持つためである．他の関数では 2 倍前後の速度向上になっているが，FPU 命令と SSE 命令のスループットの違いであり，SSE 命令により加減算や乗算で 4 並列演算をするときは FPU 命令のスループットの半分であることに起因する．

3D Now! 命令を実行できるプロセッサでは FPU 命令とスループットやレイテンシはほぼ同じであるため，表 2 に示されるようにおおむね 2 倍前後の結果になっていることは，3D Now! 命令が 2 並列処理であることを考えれば妥当である．0.75 乗演算が 3 倍超の速度向上を得ているが，これは SSE 命令の rsqrt 命令相当の命令が，一連の複数の命令に分割されていることによるスループットの低下はあるものの，一連の複数の命令が FPU 命令よりもスループットが高いことによる．

並列実行時では，共有メモリを使用しない関数では実行時間の相違は小さかったが，エンコード開始時に 1 フレーム分の PCM データをバッファ（プロセッサ間の共有メモリ）にシフトインする部分は大きな相違が見られた．**表 3** に結果を示す．キャッシュ制御命令を使用しているが，主記憶へのアクセスはキャッシュからのアクセスに比較して遅い．

表 3 関数ごとの実行時間（Intel PentiumIII 700 MHz Dual）
Table 3 Elapsed clocks for each function (Intel PentiumIII 700 MHz Dual).

	1 CPU $[\times 10^3 clk]$	2 CPUs $[\times 10^3 clk]$	Speed-up
PCM shift-in	69.8	43.6	0.16

5. おわりに

IA-32 をターゲットに MP3 エンコーダ LAME を高速化した「午後のこ〜だ」はネットで公開した結果，広く使われるようになった．最終的なバージョン 2.39 におけるソースファイル行数は，C 言語記述がコメント込で合計約 14,000 行，アセンブリ記述が約 18,000 行となっている．プロファイルをとった結果では，C 言語のままで残っている部分の実行時間は心理解析ありのエンコードで約 36%，同心理解析なしでは約 15%と残り少なくなっている．このような高速化は有用であると思われるので，人手を使用しないで行えるようにすべきであり，将来の課題としたい．

実験では「午後のこ〜だ」Version 2 について述べたが，現在は「午後のこ〜だ」Version 3 を実装中であり，実装途中ながら配付を開始している．Version 3 では，SSE2 命令による高速化や，AMD Athlon MP® プロセッサへ対応しているところである．LAME の改善に追随するように，Version 2 では実装が不十分だった VBR や新たに ABR エンコーディングもサポートするようになった．メーカからの資料も増え，「午後のこ〜だ」Version 2 では足りない部分も実装されたため，改めて人手で高速化し評価する予定である．

謝辞 LGPL に基づき MP3 エンコーダ LAME を開発している方々[2]，「午後べんち」への参加者[4]，NASM を開発している方々へ[12]，深く感謝いたします．

参考文献

1) ISO/IEC 11172-3: Coding of Moving Pictures and Associated Audio for Digital Strage Media at up to about 1.5 Mb/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) 午後べんち．http://homepage1.nifty.com/herumi/gogo_ bench.html
5) 酒居敬一，金蔵善紀，小松斉一，石本健一，阿江 忠：VLIW プロセッサ SBC について，信学技報，CPSY98–48, pp.1–8 (1998).
6) Sakai, K., Fujiwara, I. and Ae, T.: Extended

VLIW Processor for Real-Time Imaging, Real-Time Imaging V, *Proc. SPIE*, Vol.4303, pp.43–50 (2001).
7) 中田育男：コンパイラの構成と最適化，朝倉書店 (1999).
8) *AMD Extensions to the 3D Now! and MMX Instruction Sets Manual*, AMD (2000).
9) *AMD Athlon Processor x86 Code Optimization Guide*, AMD (2000).
10) *The Intel Architecture Software Developers Manual*, Volume 1–3, Intel (1999).
11) *Intel Architecture Optimization Reference Manual*, Intel (1999).
12) Netwide Assembler, version 0.98 (1999). http://www.web-sites.co.uk/nasm/
13) Intel Compiler. http://developer.intel.com/software/products/compilers/
14) 酒居敬一，光成滋生，成田 剛，石田 計，藤井寛，庄司信利：MP3エンコーダの高速化，情報処理学会研究報告，2001-ARC-144-25, pp.141–146 (2001).
15) Goldstein, S.C., Schmit, H., Budiu, M., Cadambi, S., Moe, M. and Raylor, R.R.: PipeRench: A Reconfigurable Architecture and Compiler, *IEEE Computer*, pp.70–77 (2000).
16) 五島正裕，グェンハイハー，縣 亮慶，中島康彦，森眞一郎，北村俊明，富田眞治：Dualflow アーキテクチャの命令発行機構，情報処理学会論文誌，Vol.42, No.4, pp.652–662 (2001).
17) 松崎秀則，藤井寛子，近藤伸宏：高速に out-of-order 実行するためのレジスタ割当て手法，情報処理学会論文誌，Vol.42, No.4, pp.838–846 (2001).
18) 小林良太郎，山田祐司，安藤秀樹，島田俊夫：2レベル表方式による分岐先バッファ，情報処理学会論文誌，Vol.41, No.5, pp.1351–1359 (2000).
19) 田中良夫，松田元彦，久保田和人，佐藤三久：COMPaS: Pentium Pro を用いた SMP クラスタとその評価，情報処理学会論文誌，Vol.40, No.5, pp.2215–2224 (1999).

(平成 13 年 8 月 31 日受付)
(平成 14 年 2 月 13 日採録)

酒居 敬一（正会員）
1970 年生．1993 年広島大学工学部卒業．1995 年同大学院工学研究科博士課程前期修了．同年広島大学工学部助手．計算機アーキテクチャ，コード最適化に興味を持つ．

光成 滋生
1972 年生．1995 年京都大学理学部卒業．1997 年同大学院理学研究科数学・数理解析専攻博士課程前期修了．現在株式会社ピクセラ勤務．楕円曲線暗号を研究．

成田 剛
1976 年生．1999 年弘前大学理工学部電子情報システム工学科卒業．同年株式会社インテリジェントシステムズ勤務．コンシューマ向けゲームソフトのプログラマ．

石田 計
1967 年生．1990 年筑波大学中退，同年株式会社ミクニ勤務，1994 年フォトクラブサムス経営．1996 年法人化（有限会社サムス）．画像と音声の圧縮の最適化に興味を持つ．

藤井 寛
1978 年生．2001 年岡山大学工学部卒業．現在同大学院自然科学研究科在学．情報計測・認識技術および獲得した情報の表現に関する研究に従事．

庄司 信利 現在株式会社コニカ勤務．