卒業論文

PC クラスタ上での AAC エンコーダの並列化

氏　　名：亀井 雄介
学籍番号：2260040024-3
指導教員：山崎 勝弘　教授
提 出 日：2008 年 2 月 20 日

立命館大学理工学部電子情報デザイン学科

内容梗概

並列処理を行うことによって、大規模な計算の処理時間を短縮することが可能となる。計算時間がかかりすぎる地球規模での気象予測などの大規模な計算が PC クラスタを使用することで、現実的な時間で計算が行うことができ、実用化されている。また、大規模な計算機でしかできなかった並列処理が、現在では PC 一台でも行うことができる環境になってきた。

現在、PC を使っての音声編集が一般的になってきており、音声圧縮方式において、AAC が普及してきている。本研究では、OpenMP による AAC エンコーダの並列化を行い、エンコード時間の高速化を目的としている。実験では、AAC エンコーダにおける、MDCT、符号化ツール群、量子化器、ハフマン符号化部、ビットストリーム形成部という、エンコード処理のほぼ全ての部分を OpenMP により並列化し、実行プロセッサ数が 4 で約 3.8 倍という理想的な速度向上が得られた。

目次

図目次

表目次

1. はじめに

コンピュータにおいて複数のプロセッサで 1 つのタスクを動作させる事を並列処理と言う。PC クラスタを用いた並列処理を行うと、大規模な計算の大幅な時間短縮が可能となる。PC クラスタとは複数台の PC を組み合わせ、ひとまとまりのシステムにしたものである。並列処理を行うには、複数台の PC をネットワークで接続し、さらに、その複数台の PC を一つのシステムのように扱えるようにする PC クラスタ専用のソフトウェアと、PC クラスタに計算させるための並列プログラムが必要である。これらの要件を満たして、PC クラスタは構成要素である複数の PC を一つのシステムのように扱うことが出来き、並列処理を行うことができる。

近年、PC クラスタの構成要素となる PC 自体の性能が近年大きく向上しているため、PC クラスタの性能は著しく向上している。例えば、今まででは計算に時間がかかりすぎて実現不可能と言われていた、地球規模での気象予測、DNA 解析などが、現在では PC クラスタで実現されている。PC クラスタ自体の性能も向上しており、時間がかかりすぎて不可能な計算、将来的には可能になるであろう。

このようにPCクラスタは、主にプログラムの高速化のために利用され、主なメモリモデルとして次の３種類が存在する。プログラムは並列処理を行っている全てのPCのメモリにアクセスすることができ、物理メモリを共有して処理を行うSMP(共有メモリモデル)。プロセッサは他のプロセッサの主記憶の読み書きを行うことができる分散共有メモリモデル。プロセッサは他のプロセッサの主記憶を読み書きすることはできない分散メモリモデルの３種である。

また、並列プログラミングには大きく分けて２つあり、分散メモリプログラミングと共有メモリプログラミングである。分散メモリプログラミングにはメッセージパッシング式のMPIやPVM、共有メモリプログラミングにはfork-join式のOpenMPを用いる事ができる。

本研究では、AAC エンコーダを対象に並列処理を行って、エンコーダの処理時間の向上を図っている。近年、PC はもちろん、携帯デジタル音楽プレイヤーや携帯電話でも、音楽を聴ける機器が大幅に増えてきている。ここで注目されている音声圧縮方式が、「AAC」である。AAC（MPEG-2/AAC：Advanced Audio Coding）は従来の音声圧縮方式よりも高音質・高圧縮をともに両立しており、地上デジタル放送・BS デジタル放送の音声、携帯デジタルオーディオプレイヤー、PC のメディアプレイヤーをはじめとする、多くの機器や、ソフトウェアでの利用が増加している。音声は信号として伝達する際、エンコードしデジタル化された状態で送り出される。このエンコードは一瞬で出来ず時間がかかるため、短時間でのエンコードが望まれる。本研究での AAC エンコードには、本研究室が所持する Diplo クラスタを使用する。

本論文では、本章で研究全体の背景と目的を明らかにし、第２章で AAC のアルゴリズムを述べる。第 3 章では、リファレンスエンコーダの解析について述べる。第 4 章では、AAC

の並列化アルゴリズムと OpenMP による並列化手法について述べる。第 5 章では、計測したデータを元に AAC エンコーダの並列化における実験結果と考察について報告する。

2. AAC エンコーダ

2.1 AAC とは

AAC(Advanced Audio Coding) とは MPEG の音声圧縮規格の一つで、MPEG-2/AAC, MPEG-4/AAC として、MPEG(Moving Picture Experts Group)が開発した音声フォーマットである。その特徴は、高音質・高圧縮率を両立できることであり、特に低ビットレート（高圧縮率）の環境において、MP3(MPEG Layer-3) などの従来のフォーマットに比べて、高い音質を維持することができる。これは、すでに国際標準化が完了していた MPEG オーディオの互換性よりも、性能を最優先して標準化されたからである。互換性を重視すると、制約が多く、音声改善に限界が生じるので、AAC はこれらに縛られないよう互換性を排除したものとなっている[2][13]。

AAC は用途に応じて、組み込む技術の組合せを選び、圧縮する。プロファイルは 3 種類あり、LC(Low Complexity) プロファイル、Main プロファイル、SSR（Scalable Sampling Rate）プロファイルがある。LC プロファイルは簡易版で低演算型であり、メモリや CPU の利用効率が良く、最も利用されている。Main プロファイルは音質重視型で、SSR プロファイルはサンプリング周波数と帯域を階層化したプロファイルである。

AAC は地上デジタル TV 放送、BS デジタル放送、携帯電話、メディアプレイヤー、携帯音楽プレイヤーなどの多くの機器やソフトウェアで利用されており、これらでは現在、LC プロファイルが利用されている。

エンコーダとは、データのエンコードを行うソフトウェア、またはハードウェアであり、データを一定の規則に基づいて目的に応じた符号に変換する。例えば、音声や動画をコンピュータで扱えるように符号化することである。コンピュータにおいてはデータやファイルの圧縮、あるいは暗号化のことを指す場合もある。エンコードされたデータは、デコーダで復号を行うと元のデータに戻すことができる。

2.2 AAC のアルゴリズム

図 1 に AAC エンコーダのアルゴリズムを、表 1 に図 1 における各モジュールへの入出力と処理内容を示す。処理ごとの概要は以下の通りである。

音声信号入力は、圧縮符号化する音声(オーディオ)信号である。AAC では 44.1Hz ステレオだけでなく、48kHz、96kHz の信号や、5.1 チャンネル・サラウンドなども入力として処理することができる。

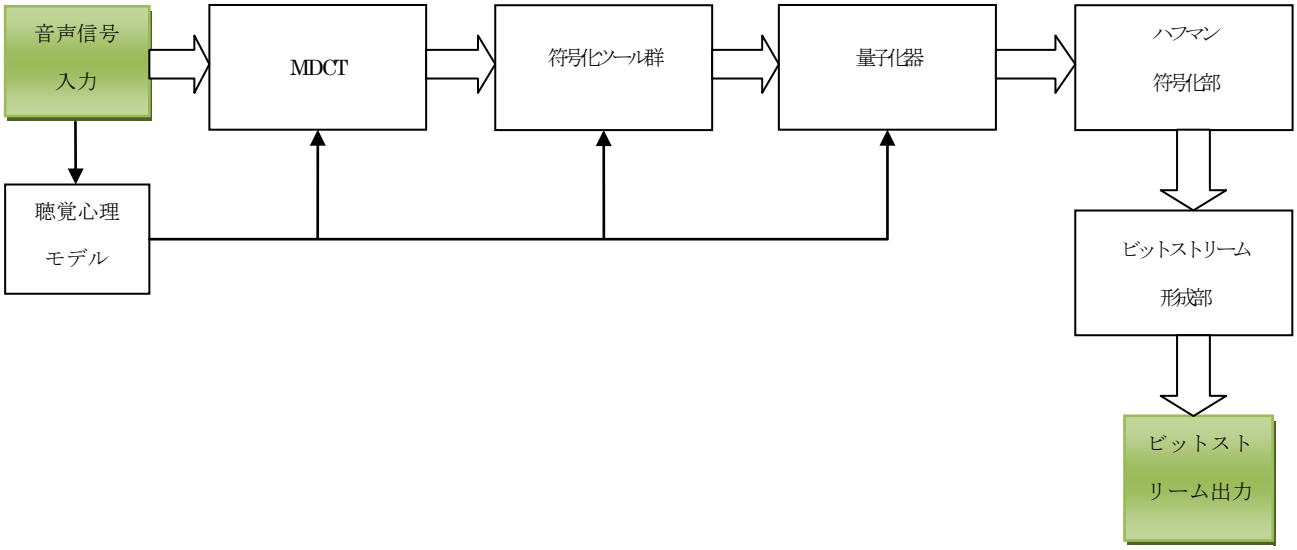

図 1：AAC のアルゴリズム

表 1：各モジュールの処理と入出力

入出力と処理内容 / モジュール	処理内容	入力	出力
MDCT	入力データを複数の周波数帯域に分割/周波数スペクトルの算出	音声信号	周波数スペクトル（MDCT 係数の列） ＜周波数域信号＞
符号化ツール群	MDCT 係数のビット数削減処理	MDCT 係数列	削減された MDCT 係数列
量子化	値を切り捨て、切り上げして整数値化	削減された MDCT 係数列	量子化された MDCT 係数列
ハフマン符号化	2 進数化	量子化された MDCT 係数列	2 進表現されたデータ
ビットストリーム形成部	フレームに様々なデータを組み込む	2 進表現されたデータ	ビットストリーム
聴覚心理モデル	各モジュールの処理における最適な値、処理方法の分析（FFT などを用いる）	音声信号	各 MDCT 係数列に対する量子化精度、マスキングしきい値の計算結果、など

(1) MDCT

MDCT では、MDCT(Modified Discrete Cosine Transform)は、修正離散コサイン変換と訳され、入力された音声信号を所定のサンプルごとに時間域信号から周波数域信号へと変換する。隣接するブロック間で 50%のオーバーラップをかけて、窓を重複させながら周波数変換を行う重複直行変換である。MDCT は変換後に逆変換したときに完全に波形が再構成される可逆変換である。また、所定のサンプルと採る際において、単位時間当たりにサンプル採る頻度をサンプリング周波数 [Hz]という。一般的な CD の音声データはサンプリング周波数 44.1kHz でサンプルが採られている。また、音声圧縮で、1 秒あたりに音声に与えるデータの容量をビットレート[bps]という。

しかし、オーバーラップすることにより、変換前の信号のサンプル数 N より変換後の信号のサンプル数 M が多くなると、符号化しなければならないデータ数が増加することになる。効率よく符号化するためには $M \leq N$ を満たすことが望ましい。

MDCT はⅣ型 DCT をベースにした変換式を持ち、以下の手順により変換および逆変換を行う。

(ⅰ)窓かけ(順変換用)

入力時間信号 x(n)に順変換用の窓 $\omega_1(n)$, $0 \leq n \leq 2M$ をかける。

$$x_{1,J}(n) = \omega_1(n)x(n + JM), \qquad 0 \leq n < 2M$$

(ⅱ)順変換

窓かけした後の信号$x_{1,J}(n)$を次式により MDCT 係数 $X_J(k)$ に変換する。

$$X_J(k) = \frac{2}{M}\sum_{n=0}^{2M-1} x_{1,J}(n)\cos\left(\frac{\pi(2k+1)(2n+M+1)}{4M}\right), \quad 0 \leq k < M$$

(ⅲ)逆変換

MDCT 係数$X_J(k)$を次式により逆変換する。

$$x_{2,J}(n) = \sum_{k=0}^{M-1} x_J(k)\cos\left(\frac{\pi(2k+1)(2n+M+1)}{4M}\right), \quad 0 \leq n < 2M$$

(ⅳ)窓かけ(逆変換用)

逆変換の信号$x_{2,J}(n)$に逆変換用の窓 $\omega_2(n)$, $0 \leq n < 2M$ をかける。

$$x_{3,J}(n) = \omega_2(n)x_{2,J}(n), \qquad 0 \leq n < 2M$$

(ⅴ)オーバーラップ

J-1 番目の $x_{3,J-1}(n)$ の後半部分と、J 番目の $x_{3,J}(n)$ の前半部分を加え合わせることにより出力時間信号 $y(n + JM)$ を得る。

$$y(n + JM) = x_{3,\ J-1}(n + M) + x_{3,J}(n), \qquad 0 \leq n < M$$

以上のように長さ 2M の窓をかけ、50%オーバーラップさせながら変換を行うことにより、M 個の実係数を得ることが分かる。すなわち、M 個の入力サンプルに対し、M 個の係数を得ることになり、符号化する数値の数は増加しない。

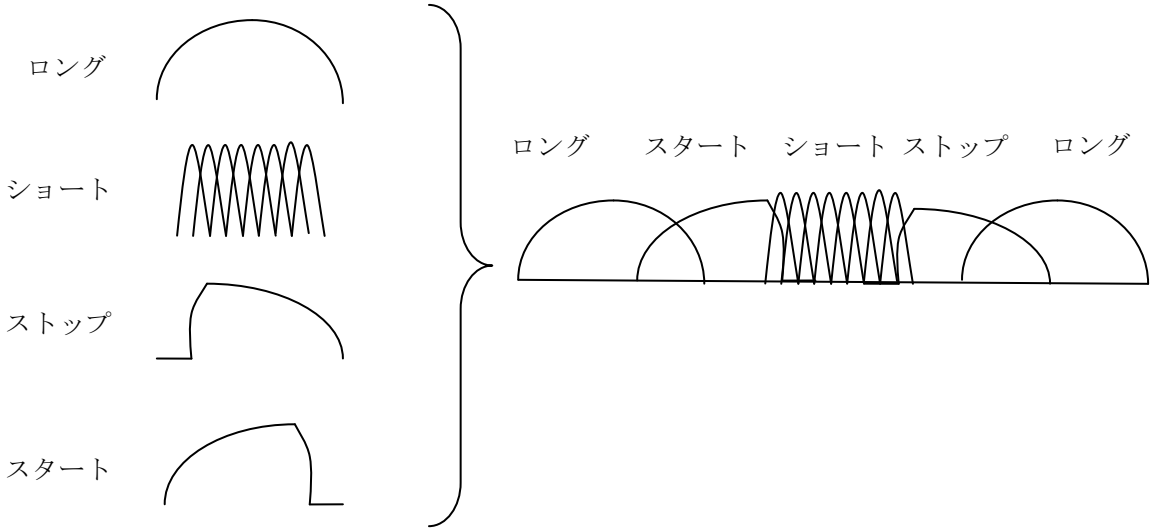

図 2：MDCT 用窓

AAC の MDCT は、1024 サンプルを 1 つのフレームとしている。MDCT は 2048 フレームの長さを 1 回、もしくは 256 フレームの長さを 8 回の変換を選択して行う。2048 フレームの長さ 1 回の MDCT(ロング)と 256 フレームの長さ 8 回の MDCT(ショート)の間を滑らかにつなぐために、スタートおよびストップという非対称の窓を用いている。非対称の窓であるがオーバーラップする部分では対象に窓がかかる。図 2 にその窓の種類と、切り替え方の例を示す。窓関数はサイン窓と Kaizer-Bessel 窓をベースに算出した窓をフレームごとに切り替えることができる[3]。

また、MDCT と DCT の変換を比べたものを図 3 に示す。図 3 のように、DCT は N 個の時間域の sample から、N 個の周波数域の sample を導出する。しかし、MDCT は 2N 個の時間域の sample から M 個の周波数域の sample を導出する。MDCT の場合は、前後の MDCT 変換に用いる sample が N 個重なっている。

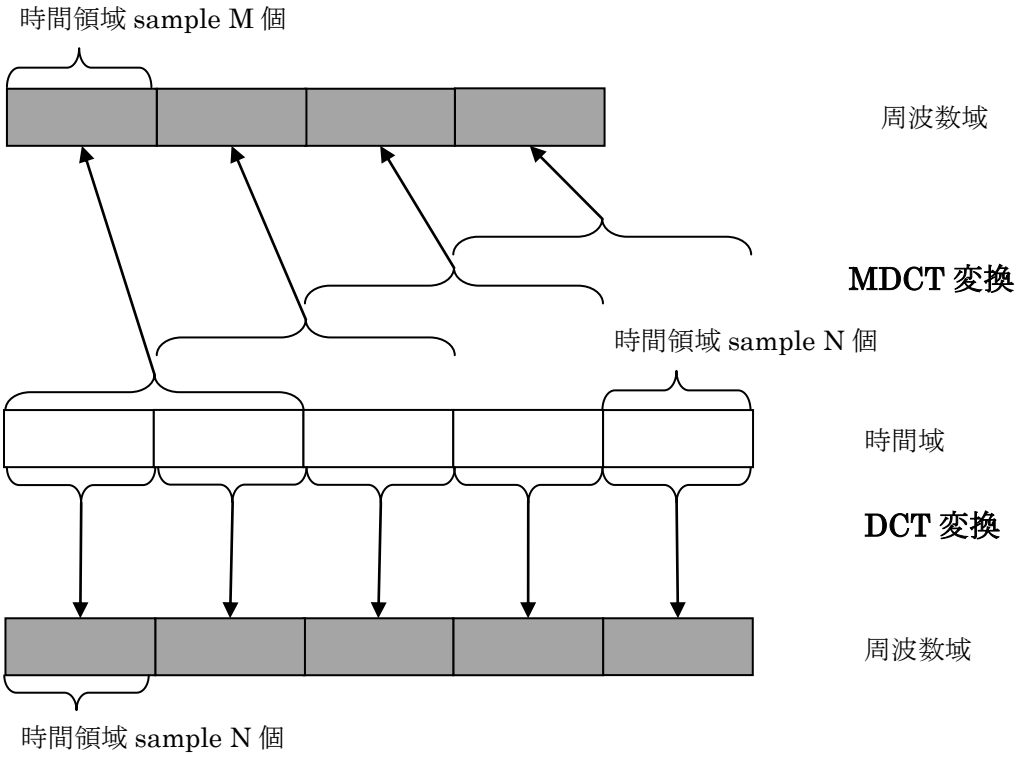

図 3 : MDCT と DCT の違い

(2) 聴覚心理モデル

聴覚心理モデルでは、人間の聴覚をシミュレートし、符号化処理を聴覚特性に合わせて最適化するための演算を行う。そのうちの一つとして、周波数マスキングが行われる。周波数マスキングとは、人間は絶対可聴しきい値よりも大きな音しか知覚できない、大きな音の周波数の近傍の小さな音の周波数は知覚できない、という聴覚の性質を利用して圧縮効率を上げる方法である。図 4 に周波数マスキングの効果を示す。

図 4 において、青い矢印は人間に聞こえる音、赤い矢印は聴こえない音である。3 つある赤い矢印の内、左の 2 つの赤い矢印で表わされる音は、人間に聞こえる周波数の範囲である、絶対可聴しきい値を下回っているので聞こえない。また、②の音は絶対可聴しきい値を上回っているが、①で示された強い青い信号によるクリティカルバンドによってマスクされて聴こえなくなる。聴覚には、大きな音と小さな音が同時にあるとき、小さな音は聞こえ難くなり、更にこれら 2 つの音の周波数が近いほど、小さな音は聞こえ難くなるという特性があり、これをマスキング効果といい、マスキング効果の及ぶ範囲をクリティカルバンドという。このようにして、聴覚的に人間に聞こえない音を省いて圧縮効率を上げる。このように人間の聴覚を利用して圧縮率の向上を行う。この事を考慮して量子化での量子化ビット数を決めるのに参考にする値を算出し、量子化器に送り出せば、より効率的に量子化を行うことができる[2]。

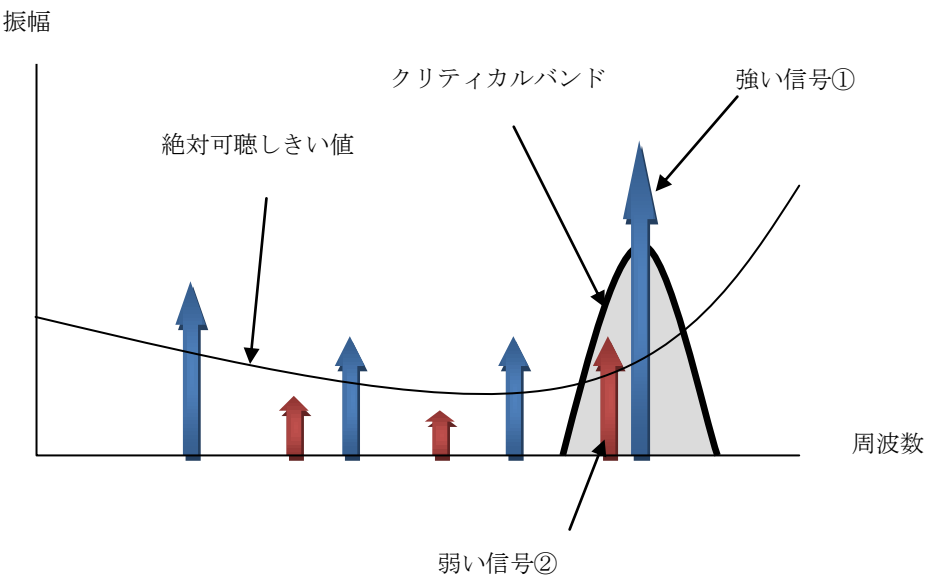

図 4：周波数マスキングの効果

このようにして、聴覚的に人間に聞こえない音を省いて圧縮効率を上げる。このように人間の聴覚を利用して圧縮率の向上を行う。この事を考慮して量子化での量子化ビット数を決めるのに参考にする値を算出し、量子化器に送り出せば、より効率的に量子化を行うことができる。

他には、各周波数帯域の MDCT 係数に対する量子化精度を求める。これらを聴覚心理モデルでは FFT を用いて計算する。FFT(Fast Fourier Transform：高速フーリエ変換)を行うことにより、離散信号を時間領域から周波数領域へ変換することができる。高速フーリエ変換とは文字通りフーリエ変換(Discrete Fourier Transform： DFT)を高速化したものである。

フーリエ変換の基本式は以下のようになる。

$$X(k) = \frac{1}{NT}\sum_{n=0}^{N-1} x(nT)e^{-j\frac{2\pi}{N}kn}, \qquad 0 \le k < N-1$$

この式を高速化させたものを FFT と呼び、基本式は以下のようになる。

$$X(k) = \frac{1}{2}\left(X_0(k) + e^{-j\frac{2\pi}{N}k}X_1(k)\right), \qquad 0 \le k \le \frac{N}{2}-1$$

$$X\left(\frac{N}{2}+k\right) = \frac{1}{2}\left(X_0(k) + e^{-j\frac{2\pi}{N}\left(\frac{N}{2}+k\right)}X_1(k)\right), \qquad 0 \le k \le \frac{N}{2}-1$$

$$= \frac{1}{2}\left(X_0(k) - e^{-j\frac{2\pi}{N}k}X_1(k)\right)$$

このように、フーリエ変換は行われる[2]。

(3) 符号化ツール群

符号化ツール群では、AAC の圧縮符号化の性能を上げる効果を持つ。これらは、聴覚や信号の性質の中で特殊な性質を使って、入力された MDCT 係数のビット数を削減し、わずかでも性能を上げるためのものである。他の処理と比べると全体の効果に占める処理はわずかである。符号化ツール群には以下のものが含まれている。

(a)TNS（Temporal Noise Shaping：時間領域量子化雑音整形）

TNS は、MDCT 部が出力したスペクトル信号（MDCT 係数）を時間軸上の信号であるかのように見立てて線形予測を行い、MDCT 係数に対して予測フィルタリングを行う。この処理により、デコーダ側で逆 MDCT して得られる波形に含まれる量子化雑音は、信号レベルの大きなところに集まるようになる。なお、TNS は予想利得がしきい値を超えたときだけ実行される。

符号化しようとしている MDCT 係数を図 5 のように TNS フィルタに入力し、フィルタ出力を量子化および符号化するとともにフィルタ係数などの補助情報も符号化する。デコーダではエンコーダと逆特性のフィルタに通すことにより、量子化された MDCT 係数を得る。フィルタ係数は時間信号の包絡成分を表しているため、MDCT 係数の量子化による量子化雑音の影響を受けないことになる[3]。

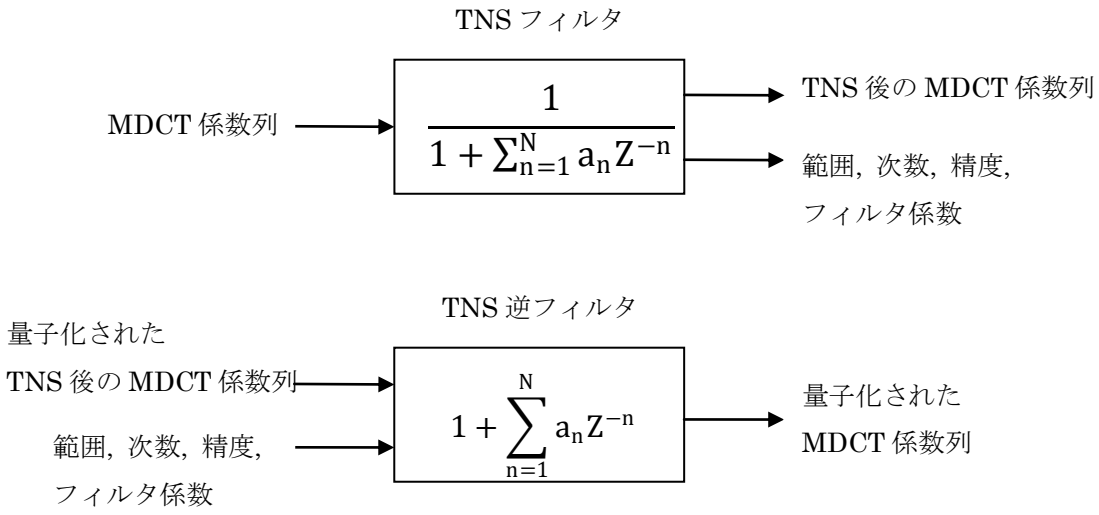

図 5：TNS

(b)インテンシティステレオ処理

インテンシティステレオ処理では、両チャネルの和信号と各チャネル信号の比を、本

来の 2 チャンネル信号の代わりに用い、音声圧縮符号化の圧縮率を向上させる[13]。

(c)予測

予測では、前の音を使って効率よく符号化する方法であり、Main プロファイルの、ショートの窓以外で使用できる。あるフレームの MDCT 係数を直前、およびその前のフレームの MDCT 係数から予測器を使って予測し、実際の MDCT 係数の値の予測値との差分を符号化することにより、定常的な信号が入力された場合の符号化効率を上げる[3]。

(d)M/S ステレオ処理

M/S(Main/Sides)ステレオ処理では、両チャネルの和信号と差信号を、本来の 2 チャンネル信号の代わりに用いる。M/S ステレオは最も簡単な 2 点直交変換であり、両チャネルの相関が大きいときには、得られた和信号と差信号との情報差が大きくなり、エネルギー偏在によるデータ圧縮効果が期待できる[13]。

(4) 量子化器

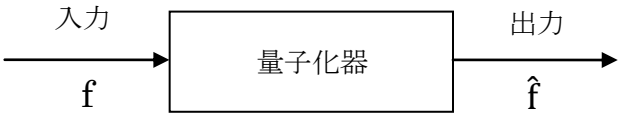

図 6：量子化器

量子化とは図 6 のように、入力信号を有限長のデジタル値に変換することである。振幅方向を離散化して周波数スペクトルの形状を量子化し、入力データの切り捨て、切り上げを行う。その方法を図 7 に示す。横軸はアナログ量の実数の入力値、縦軸は量子化された値の整数の出力値である。図 7 のように、$1.5 \leqq x < 2.5$ を満足する実数 x は、量子化により 2 になる。この量子化ステップを無限に細かく取れば、入力と出力の波形は等しくなる。聴覚心理モデルがビットレートと人間の聴覚に合わせて、最適な量子化の粗さを算出するので、それを参考に量子化を行う。連続する信号の値を離散的なデジタル値にするので、入力値と量子化値との間に量子化誤差が生まれる。よって量子化は情報の欠落は不回避であり、非可逆圧縮である。

また、AAC で行われる量子化は非線形量子化であり、量子化ステップ幅を等分にしない方法である。人間の知覚が敏感なため重要視するべき部分ではステップ幅を小さく、そうでない部分はステップ幅を大きく設定した量子化方式である[12]。非線形量子化の例を図 8 に示す。

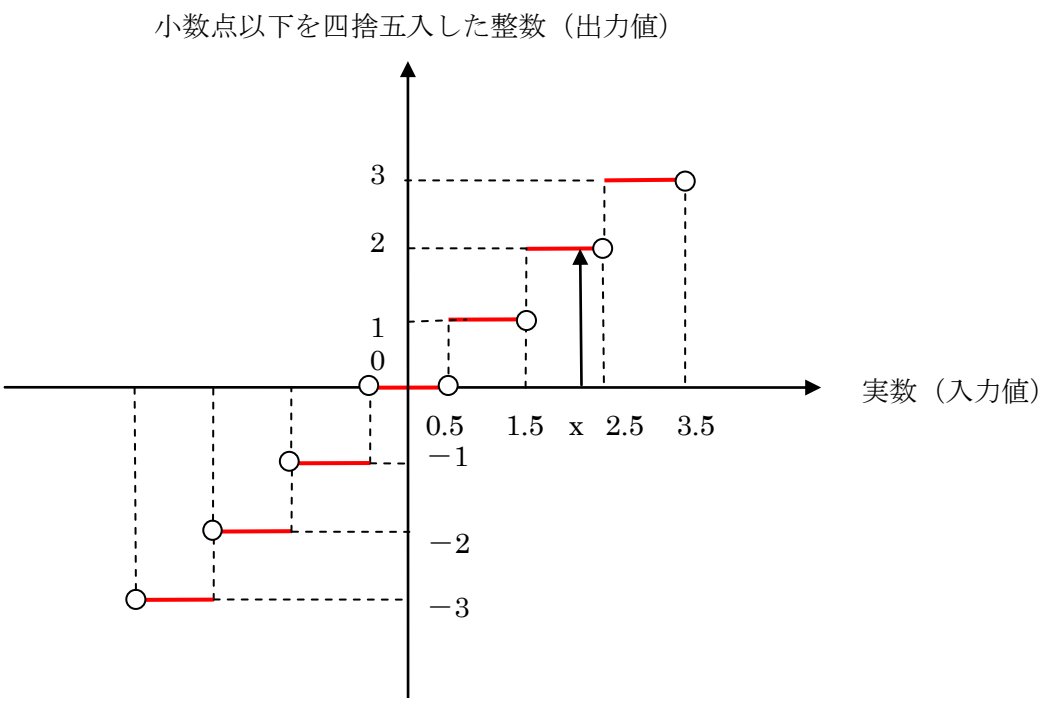

図 7： 量子化におけるデータの切り捨て、切り上げ

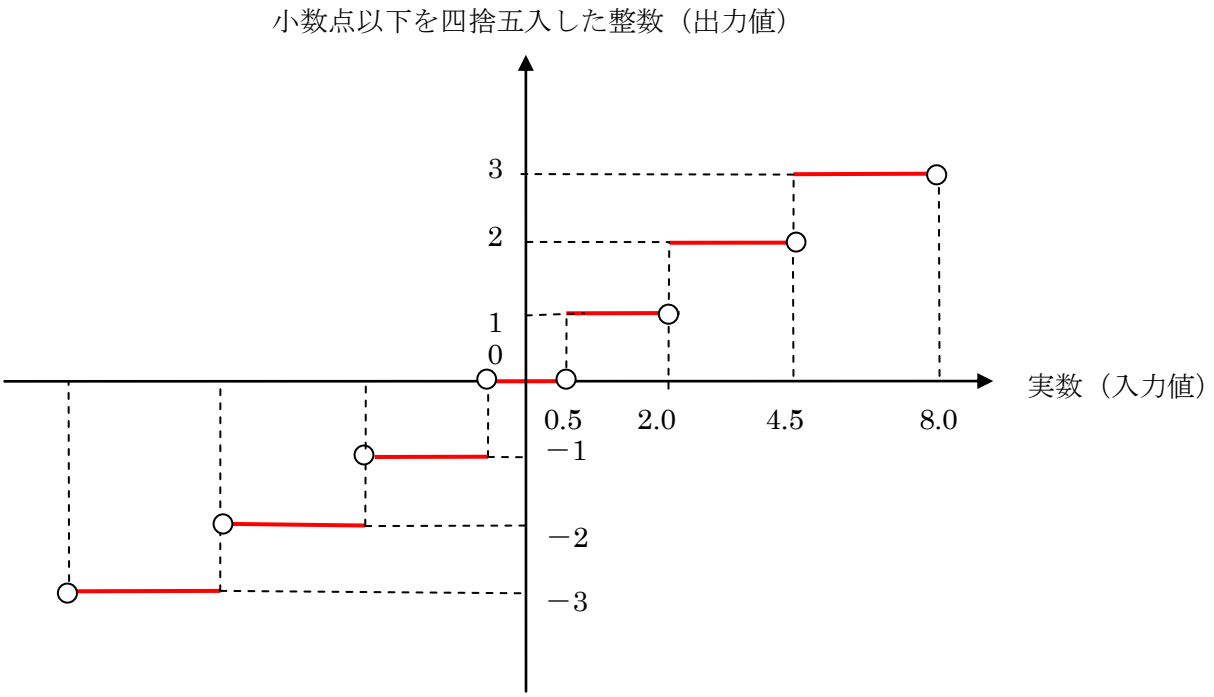

図 8： 非線形量子化の例

(5) ハフマン符号化部

ハフマン符号化は、可逆圧縮の代表的なアルゴリズムである。一定ビットごとに文字列を区切り、区切られた後の文字列を統計的に処理して、出現確率がより高いパターンに対して、より短い符号を与える方法である。各パターンの出現数を比較して、最も出現数の低いパターン 2 つに「0」と「1」の符号を付与し、次の比較においては両者を合算して扱うことにより、一意な情報を維持することを可能にしている。

ハフマン符号化はテーブルの参照で行われるので、量子化後の値がハフマンテーブルに用意されているものより大きければ、ハフマン符号化は行えないので、値が小さくなるまで量子化を繰り返す。

(6) ビットストリーム形成部

ビットストリーム形成部では、ハフマン符号化部で圧縮されたデータに、サイド情報やサンプリング周波数などのデコードする時に必要な様々なデータを組み込む。サイド情報とは、MDCT の変換ブロック長に対する情報、量子化ステップサイズ、ハフマン符号化の領域・テーブルなどに関する情報である。

3. リファレンスエンコーダの解析

3.1 ソースコード

並列化をするにあたり、AAC エンコーダを一から作成するのは大変なので、フリーソフトのエンコーダのライブラリである、FAAC - 1.26 を書き換えて並列化を行った。

3.2 各モジュールの処理時間

この FAAC エンコーダは AAC の LC プロファイルのみをサポートしている。

エンコーダのソースファイルに、処理に時間を計測するプログラムを書き加え、各モジュールでの処理にかかる時間を計測した。図 9 に時間計測の結果を示す。これは各モジュールの主要関数をループする回数から時間を計測したものである。

時間計測の際のエンコードは、Dual Intel Xeon 3GHz × 2 , 4GB SDRAM , 250GB HDD の環境で行い、入力した wav ファイルは 39369 KB で、3 分 48 秒の音声ファイルである。この音声ファイルの逐次でのエンコードには 14.53 秒かかる。

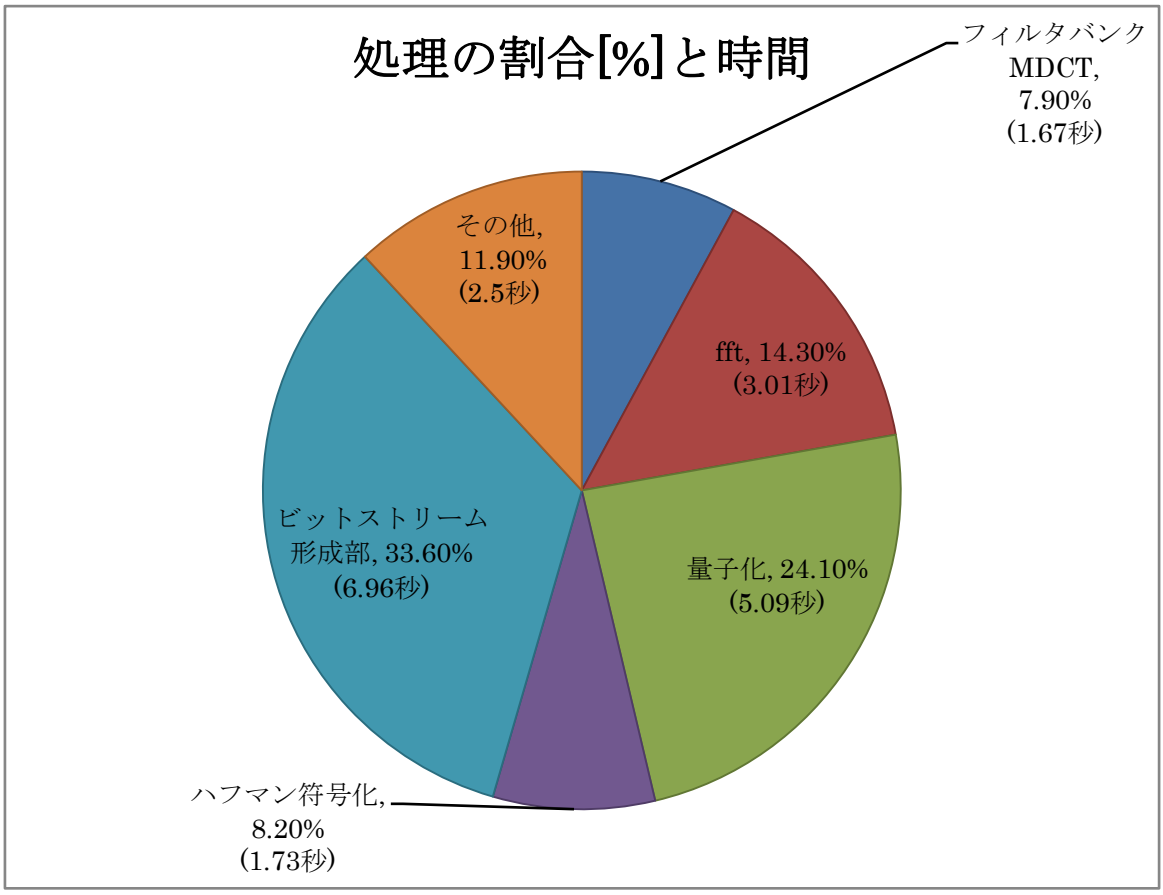

図 9：各モジュールの処理時間と割合

この結果より、ビットストリーム形成部が最も処理時間を要していて、このモジュールを並列化すると成果が得られると考えられる。本研究では、ビットストリーム形成部だけでなく、MDCT、符号化ツール群、量子化器、ハフマン符号化部、ビットストリーム形成部を並列化する。

4. AAC エンコーダの並列化

4.1 AAC エンコーダ並列化アルゴリズム

本研究に使用している AAC エンコーダは 2-2 節の通り、音声信号入力から、MDCT、符号化ツール群、量子化器、ハフマン符号化部、ビットストリーム形成部を経て出力される。3-2 節で述べたようにこれらの箇所を並列化する。

入力された音声データは複数のフレームに分けられて処理されていくので、そのフレームを分割し、OpenMP による並列処理を行う。並列処理において、データの分割方法には、代表的なものとして、ブロック分割と、サイクリック分割の 2 つの分割方法が存在する。本研究ではブロック分割を行ってエンコーダの並列化を行う。次に、実行プロセッサ（スレッド）数が 4 の場合での、ブロック分割方法を図 10 に、サイクリック分割方法を図 11 に示す。

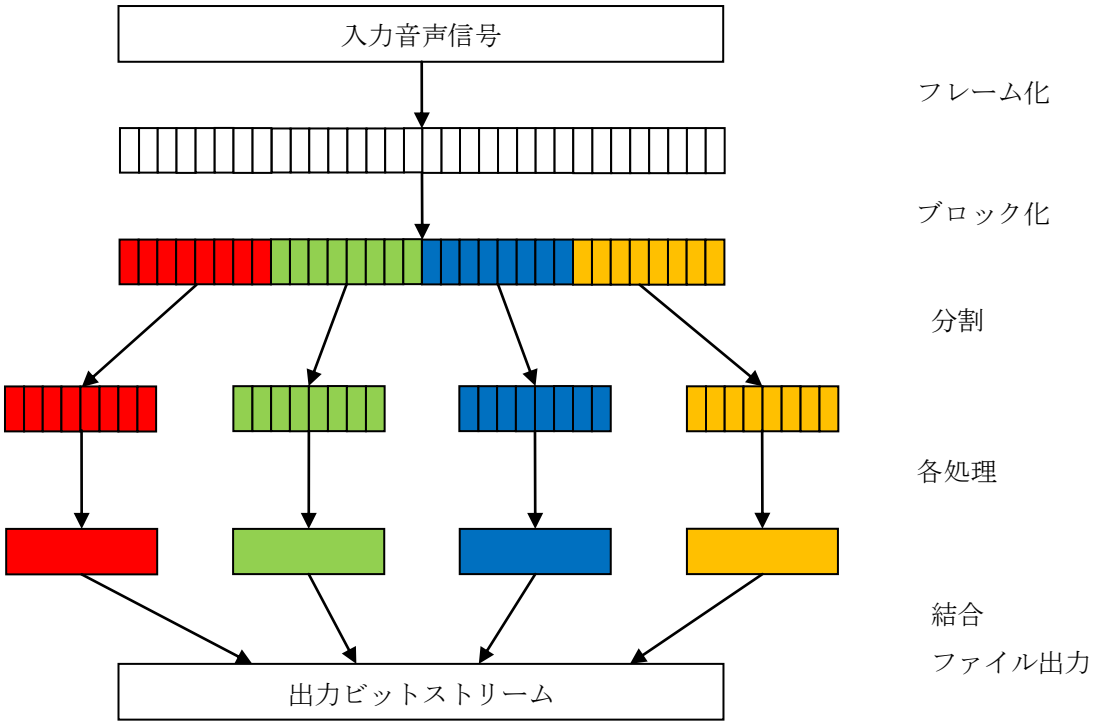

図 10：ブロック分割

ブロック分割は、図 10 のように、計算領域をあるまとまった数ごとに分割する手法である。通常はブロックがプロセッサ数になるように、均等に分割する。例えば、プロセス数が N 個の場合、各プロセスに 1/N の領域を割り当てる手法である。ブロック分割はサーバとノードの通信回数が 2 回だけですむ。

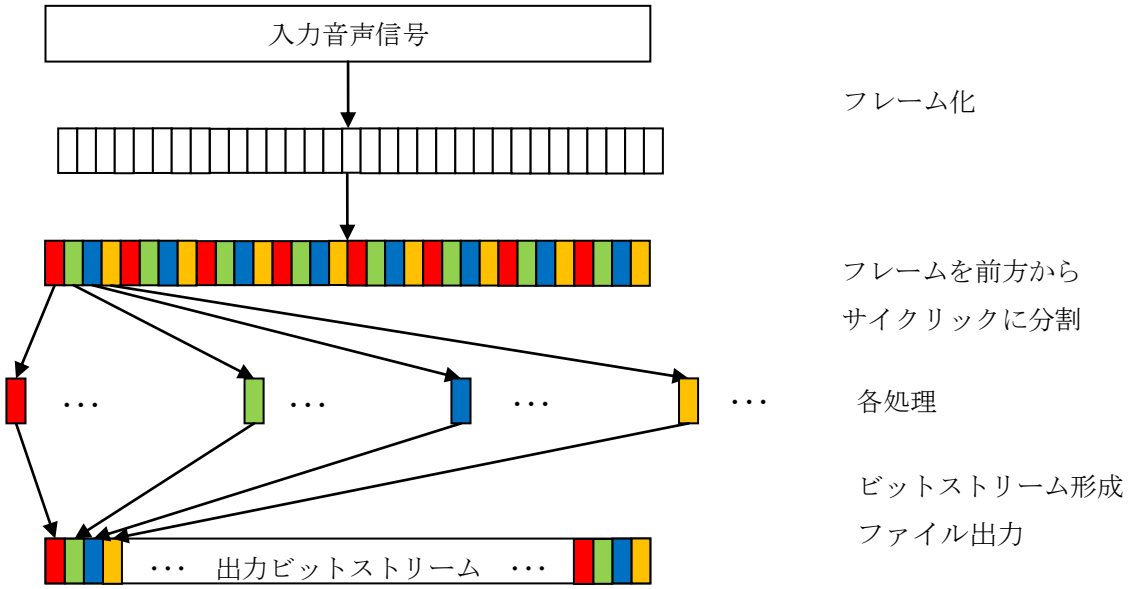

図 11：サイクリック分割

サイクリック分割は、図 11 のように、各スレッドに対して要素を一つずつ順番に、すなわちサイクリックに割り当てる手法である。よって、サーバとノードの通信回数が多くってしまう。

各計算領域のプロセスの負荷がほぼ均等な場合は、ブロック分割が望ましい。並列化において、全ての処理範囲が同じ処理量であれば、ブロック分割は効率が良い手法である。また、分散メモリ型でプロセッサ間通信が必要な場合を考慮するときも通信に要するコストを必要最小限に抑えるためには、ブロック分割が適当である。これに対して、各計算領域の負荷が不均等になるときは、サイクリック分割が適当である。

本研究において、エンコーダに入力するのは音声ファイルであり、音声ファイルをフレームに分けた場合、各フレームの負荷は、ほぼ均等になるといえる。よってブッロク分割が有効と考えられる。

4.2　OpenMP による並列化手法

まず、OpenMP について説明する。OpenMP とは、共有メモリのマルチプロセッサ環境を利用するのに用いられる API（Application Programming Interface：アプリケーション プログラミング インターフェース）であり、ベースとなる言語(Fortran, C/C++)に指示文(ディレクティブ)を追加することで並列プログラミングを行う。OpenMP は新しい言語ではなく、コンパイラ指示文、ライブラリ、環境変数によりベース言語を拡張するためのものである。また、自動並列化ではなく、並列実行・同期をプログラマが明示することで並列化を行う。

OpenMP は次のような特徴を持つ。

・既存の逐次型プログラムをベースに、並列プログラムを作成可能。
・基本的には指示文の挿入のみで、スレッドを生成・制御可能。
・並列実行部をプログラマが指示、指示文を無視すれば逐次実行も可能。
・徐々に指示文を加えることにより、段階的に並列化することが可能。
・分散されたメモリを 1 つの共有メモリとして扱えるため、データの受け渡しを考慮する必要がない。

OpenMP は fork-join モデルであり、複数のスレッド(プロセッサ)が並列プログラムを実行するとき、逐次部分は単一のスレッドで実行し、並列化された部分は他のスレッドを生成して並列処理を行い、並列指示文が終わると単一のスレッドのみに逐次実行に戻る。また、OpenMP は図 12 のように、プログラム中にあるノードの数により fork-join を繰り返す。fork-join モデルとは、複数のスレッド（プロセッサ）が並列プログラムを実行するとき、逐次部分は単一のスレッド（マスタースレッド）で実行して、プログラムが並列指示文によって並列化された部分になると、ほかのスレッドを生成(fork)して並列処理を行う。並列指示文が終わると、単一スレッドのみの逐次実行（join）に戻る。このように、プログラム中にあるノードの数により fork-join を繰り返す。また、OpenMP において、逐次処理部を逐次リージョン（Sequential region)、並列処理部を並列リジョン（Parallel region）と呼ぶ。

C 言語で OpenMP を用いて並列処理を行うときは、並列化したい部分に次のような OpenMP の指示文を挿入する。

#pragma omp *directive_name*(*clause, clause,・・・*)
Directive_name：指示文　　*clause*：指示節

このような指示文をプログラマが明示的に並列性を指示し挿入することによって、逐次プログラムから段階的・シームレスに並列化が可能となる。

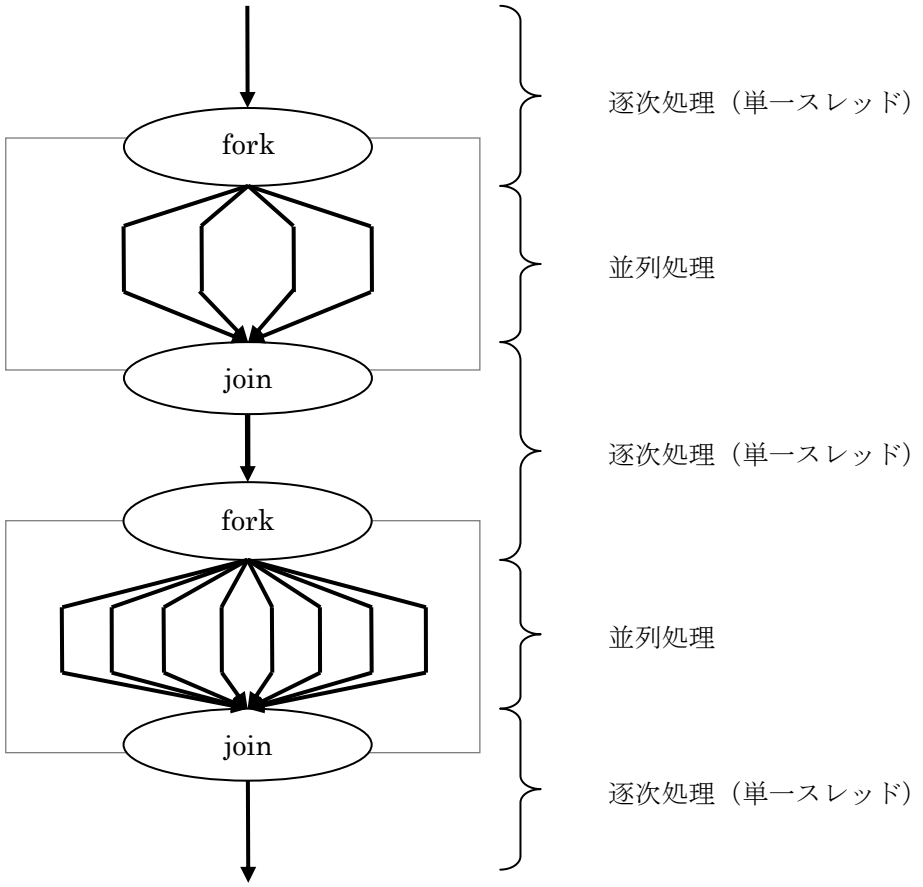

図 12：OpenMP の fork-join モデル

並列化手法についてだが、まず、本実験で使用する PC で OpenMP が使えるように、ソースコードの Makefile を次のように書き換えた。

CC = icc
CCDEPMODE = depmode=icc
CFLAGS = -openmp -2 -Wall
CPP = icc –E
CPPFLAGS =
CXX = icpc
CXXCPP = icpc -E
CXXDEPMODE = depmode=icc
CXXFLAGS = -g -O2 -xP -ipo -parallel -openmp -i-static
LDFLAGS = -static-libcxa -openmp -i-static
ac_ct_CC = icc

ac_ct_CXX = icpc

書き換えを行った後、Intel Compiler で OpenMP を用いてのコンパイル・実行が可能な事を確認した。

また、ソースプログラムのエンコードの主要実行部分に次のように OpenMP の指示文を挿入した。

```
#pragma omp parallel
{
        #pragma omp for private( )
        for ( ){
                実行部分
        }
}
```

このように Makefile と、ソースプログラム実行部分に#pragma を加え OpenMP に対応し、並列化を行える用に書き換え、Intel Compiler でコンパイルし、並列化を行った。

5. 実験と考察

5.1 実験環境

実験に用いた Diplo クラスタの構成を図 13 に示す。Dilplo クラスタは、本研究室が所持するクラスタであり、Diplo クラスタの計算ノードは、Diplo00 から Diplo03 までの計 4 台で、それぞれのノードに 2 個の Dual Intel Xeon プロセッサが搭載されている。つまり、1 台の計算ノードに 4 個のプロセッサが搭載されている。

Diplo 一台のスペックは、Dual Intel Xeon 3GHz × 2 台 ，4GB SDRAM ，250GB HDD であり、これらには Intel C/C++ Compiler がインストールされている。

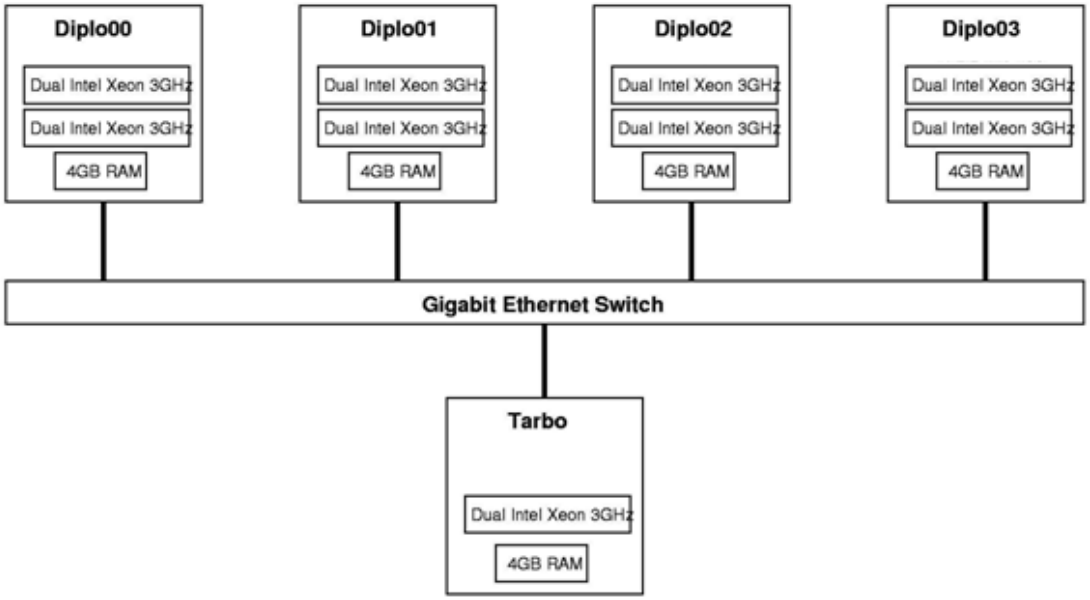

図 13：Diplo クラスタの構成

本研究では、マルチコア環境である Diplo 一台で、実行するプロセッサ数を変えて実験を行った。

5.2 実験結果

エンコードしたファイルは一般的な音楽ファイルの容量である 39369 KB で、3 分 48 秒の wav ファイルである。尚、実験は実行台数（スレッド数)、エンコードするビットレートを変えて行った。エンコードする際のサンプリング周波数は、14410Hz で行った。

本研究の実験結果を表 2 に、実行時間のグラフを図 14 に、速度向上比のグラフを図 15 に示す。

表 2： 実験結果

	実行プロセッサ数	1	2	4
128kbps	実行時間［s］	14.5310	7.4232	3.8193
	速度向上比	1	1.96	3.80
196kbps	実行時間［s］	16.2200	8.1864	4.2368
	速度向上比	1	1.98	3.83

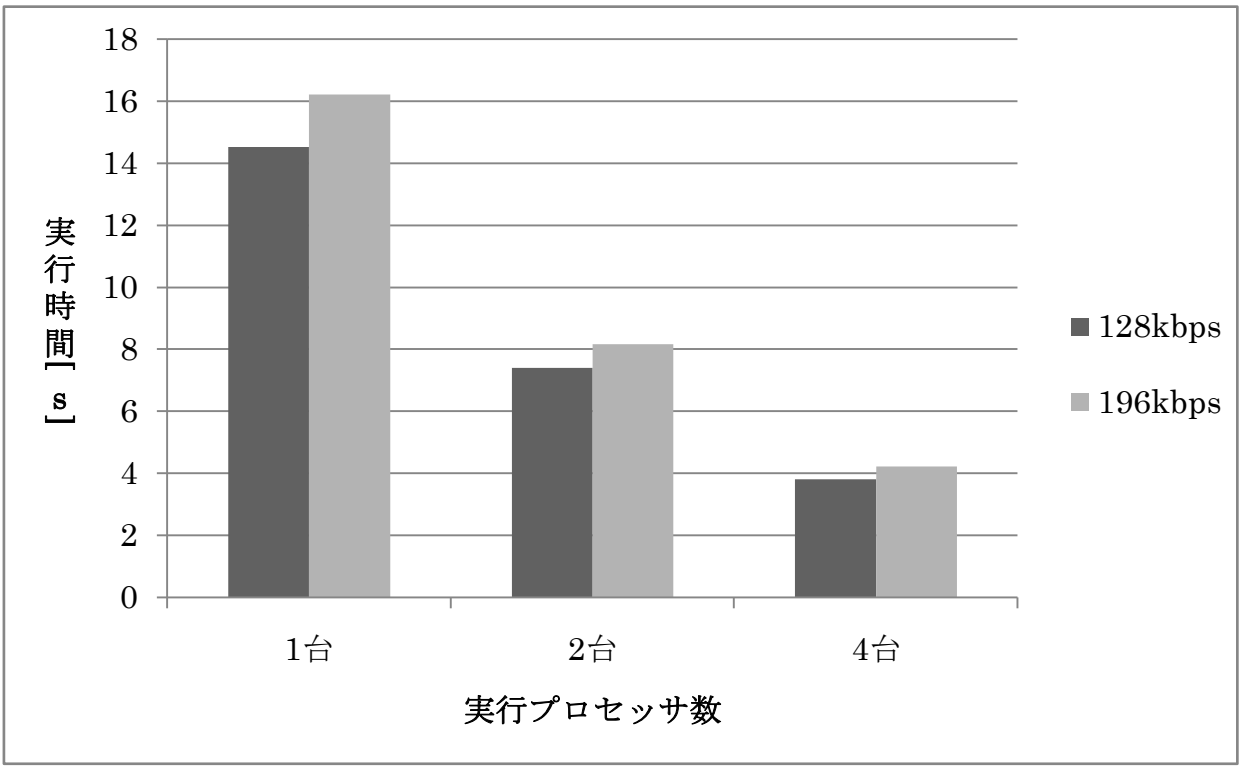

図 14： 実験結果（実行時間）

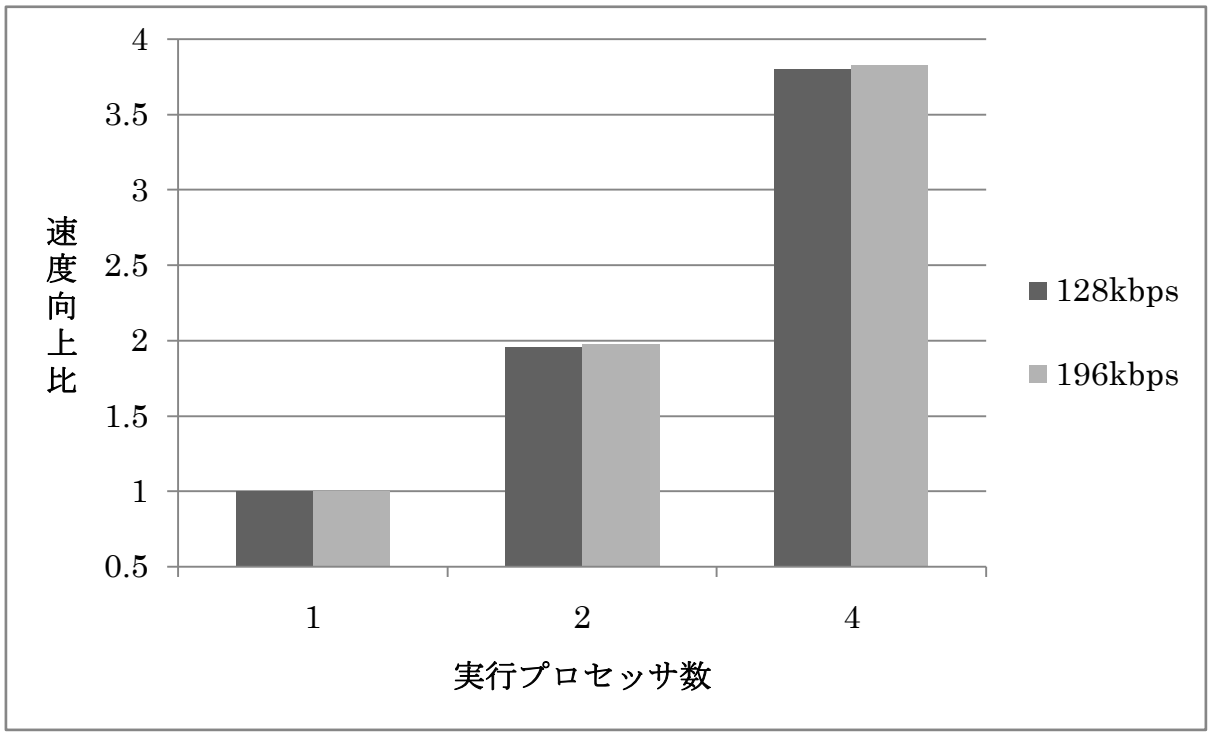

図 15：実験結果（速度向上比）

これらの実験結果より、実行プロセッサ数を増やすと、実行プロセッサ数を 2 台にすると速度向上比は約 2 倍に、4 台にすると約 4 倍となった。

5.3 考察

実験結果は、ビットレートが 128kbps で、プロセッサ数が 2 だと 1.96 倍に、プロセッサ数が 4 だと 3.80 倍に、ビットレートが 196kbps で、プロセッサ数が 2 だと 1.98 倍に、プロセッサ数が 4 だと 3.83 倍となった。この速度向上比はほぼ理想的であり、十分な並列効果を得たといえる。この高い速度向上比を得られた理由として、2 つの理由が挙げられる。1 つ目は、エンコード全体の処理の内、ほぼ全ての処理を並列化していることである。2 つ目は、エンコード処理に対して、通信の割合が少ないことである。ブロック分割は、通信回数自体が少なく、各プロセッサとの通信回数も 2 回ずつなので、エンコードの並列化には適切なデータ分割方法であった。また、計算部分が大きいため、相対的に通信部分を占める割合を少なくすることができた。

6. おわりに

今回の研究では、PC クラスタ上で、共有メモリ環境での OpenMP による AAC エンコーダの並列化を行った。実行台数を変えて、マルチコア CPU 環境で実行時間を測定し、実行プロセッサ数が 4 のときに、3.83 倍のほぼ理想的な速度向上が得られた。

また、エンコーダの並列化を行うことにより、並列プログラムの作成や、並列実行の仕方を確認することができた。

現在普及が進んでいるマルチコアの CPU では、並列処理は必要不可欠なものとなる。並列処理を行うと、快適な音声編集環境を得ることができる。

今後の研究課題として、今回行わなかったサイクリック分割を行うことが挙げられる。今回行ったブロック分割のように、サイクリック分割においても、ビットストリーム形成部を含めてサイクリックに分割して並列処理を行うと、出力されるデータ中のフレームの順番が狂ってしまう場合がある。すなわち、出力ファイル中のデータの並びが間違っているということである。このことと、サイクリック分割は通信回数が多い、ということを踏まえた、理想的な速度向上比を得られるサイクリック分割を行うことが望ましい。また今回の実験は Diplo 1 台で行ったが、Diplo 4 台使ったクラスタでの実験も挙げられる。

謝辞

本研究の機会を与えて下さり、貴重な助言、ご指導を頂きました山崎勝弘教授に深く感謝いたします。 また, 本研究に関して様々な相談に乗って頂き、貴重なご意見を頂きました、松崎裕樹氏、和田智行氏、及び高性能計算研究室の皆様に深く感謝いたします。

参考文献

[1] 湯浅太一、安村通晃、中田登志之：bit別冊 はじめての並列プログラミング
共立出版、1998
[2] 亀山渉、金子格、渡辺裕：そこが知りたい最新技術 オーディオ・ビデオ圧縮入門，株式会社インプレス R&D，2007
[3] 藤原洋：マルチメディア情報圧縮，共立出版株式会社，2000
[4] OpenMP：http://www.openmp.org/drupal/
[5] インテルコンパイラーOpenMP 活用ガイド
http://download.intel.co.jp/jp/business/japan/pdf/526J-002.pdf
[6] Omni OpenMP Compiler：http://phase.hpcc.jp/Omni/home.ja.html
[7] 田中秀宗：PC クラスタ使用法，立命館大学高性能計算研究室資料，2006
[8] 井ノ口春寿：“PC クラスタ上での Ogg Vorbis エンコーダの並列化”，立命館大学理工学部情報学科卒業論文，2007
[9] 加藤寛暁：“SMP クラスタ上での OpenMP による MPEG2 エンコーダの並列化”，立命館大学理工学部情報学科卒業論文，2006
[10] 池上広済：“ハイブリッド並列プログラミングによる MPEG2 エンコーダの高速化”，立命館大学理工学研究科修士論文，2006
[11] FAAC-1.26FAAC のライブラリ：http:// www.audiocoding.com/faac.html
[12] 中田和男、南敏：音声・画像工学，株式会社昭晃堂，1987
[13] 藤原洋、安田浩：ポイント図解式 ブロードバンド＋モバイル 標準 MPEG 教科書，株式会社アスキー，2003