卒業研究報告書

MPI による mp3 変換の検証

情報論理工学研究室

指 導 教 員

石水 隆 助教

報告者

06-1-037-0147

村吉章太郎

近畿大学理工学部情報学科

情報メディアコース

平成 22 年 2 月 5 日提出

概要

何事においても仕事の処理時間は短い方がよく、計算機の処理は常に高速なものが求められている。高速化の方法はいくつかあげられるが、その内の一つとして複数のプロセッサを用いることによる並列処理がある。しかし、複数のプロセッサを持つ並列計算機は非常に高価であるために容易に利用することはできない。そこで、複数の計算機を用いてネットワークで繋ぎ仮想的並列計算機(Parallel Virtual Computing)とすることで安価で並列処理が可能となり、スーパーコンピュータに匹敵する程の処理速度を得ることができる。

そこで本研究では仮想並列計算環境を構築するソフトウェアの１つである MPI(Message Passing Interface)を用いた MPICH2 というソフトウェアを使用し、wav(RIFF waveform Audio Format)形式のファイルから mp3(MPEG Audio Layer 3)形式のファイルへと変換する際にどれ程の時間短縮が可能であるのかを検証する。

MPI はアルゴンヌ国際研究所[4]より無料で配布されているソフトウェアであり、MPICH の公式ページからダウンロードすることができる。

目次

1 序論

1.1 仮想並列計算(Parallel Virtual Computing)

大規模なデータを高速で処理するためには並列処理が必要となるが、一般的に並列計算機は非常に高価であるために容易に使用することができない。このため、複数の計算機をネットワークで繋ぐ事により、１台の仮想的な並列計算機とする仮想並列計算(Parallel Virtual Computing)が注目されている。仮想並列計算機を構築するソフトウェアの中には無償で提供されているものもあるため、安価で並列計算環境を構築することができる。代表的な仮想並列計算環境を構築するソフトウェアとしては、PVM(Parallel Virtual Machine)[6][7]や MPI(Message Passing Interface)[1][2][3]などがある。

1.2 PVM(Parallel Virtual Machine)

PVM は並列計算を行う為のソフトウェアである。アメリカのオークリッジ国立研究所[のメンバーが中心となって開発されたソフトで、Linux、BSD、Windows など様々の OS で動作する事や、入手方法が容易である為に広く利用されている。

PVM をインストールすると、ネットワークに接続された複数台のコンピュータを単一の計算機として利用する事ができるようになる。PVM のソフトウェアシステムの構成は大きく２つに分けられ、１つ目はデーモン、２つ目はルーチンライブラリに分けられる。

適した処理は以下の通りである。

- 内部計算付加に比べ、通信負荷の高くない処理
- 非常に負荷の高い問題を異機種間共同で処理する場合
- フォールトトレランスが必要な処理
- 地理的に離れたコンピュータを使った処理
- 分散処理

1.3 MPI(Message Passing Interface)

MPI は並列・分散プロセス間のメッセージ機能を提供する標準規格。あるいは、その実装を差す。1995 年に MPI フォーラムによって標準化されて以来、多くの実装が存在しており、コードの移植性に優れているのが特徴である。複数の CPU が情報をバイト列からなるメッセージとして送受信することで協調動作を行えるようにする。自由に使用できる実装としては MPICH などがある。ライブラリレベルでの並列化であるため、言語を問わず利用でき、プログラマが細密なチューニングを行えるというメリットがある。

適した処理は以下の通りである。

- 内部計算負荷に比べ、通信負荷の高い処理
- 均一なプロセッサによる高速処理
- 短時間に行われる処理(リアルタイム処理等)

1.4 PVM と MPI の相違点

PVM と MPI の相違点を表１に示す。

表１　PVM と MPI の相違点

	PVM	MPI
異機種間による仮想並列計算	可能	不可能
フォールトトレラントへの適合性	高い	低い
メッセージ通信能力	低速	高速かつ豊富な通信関数

異機種間による仮想並列計算機は、PVM と MPI の目的の違いによる。PVM は Heterogeneous network computing を目的として作成される。それに対して MPI は高速な並列処理のためのメッセージ変換システムとして作成される。次に、フォールトトレラントへの適合性は、PVM は仮想並列計算機にプロセッサを加えたり外したりすることができるが、MPI は基本的にできない。メッセージ通信の能力は PVM は異機種間通信などをサポートするためのオーバーヘッドがあるが、MPI は通信の高速化に重点が置かれている為、PVM に対して MPI の方が一般的に高速となる。

2　研究内容

2.1 研究目的

本研究では、MPI(Message Passing Interface)を用いた MPICH2[5]というソフトウェアを使用し、wav ファイルから mp3 ファイルへの変換において１台で変換処理を行った場合と複数台で変換処理を行った場合、どの程度の時間短縮ができたのかを検証する。

更に、変換を行う wav ファイルを均等に振り分けた場合とスペック毎にファイルのサイズを変更して振り分けた場合においても同様にどの程度処理時間に変化があるのかを検証する。

2.2 準備

2.2.1 使用ソフトと使用機器

本研究では MPI を構築するソフトウェアとして MPICH2 を用いる。そして、使用した PC の性能については表２に示す。そして図１に本研究で使用した計算機ネットワークの概念図を示す。

表２　開発環境

	使用したホスト				
ホスト名	Murayoshi	Hokazono	Kanehisa	Magician2	FM
OS	Windows XP Pro	Windows XP Pro	Windows XP Pro	Windows XP Pro	Windows XP Pro
CPU	Pentium 1.60GHz	Pentium 1.60GHz	Pentium 1.60GHz	Pentium4 2.50GHz	Pentium4 3.20GHz
RAM	512MB	512MB	512MB	0.99GB	1.99GB
振分けたファイル	100MB	100MB	100MB	300MB	600MB
	150MB	150MB	150MB	150MB	

2.2.2 MPICH-2 のインストールと環境設定

MPICH-2 を使用するために、各計算機に Windows 用の MPICH-2 のインストールを行う。MPICH-2 は、アルゴンヌ国際研究所[4]の MPICH-2 の公式ページ[5]において無償で提供されており、これをダウンロードした後各計算機にインストールする。

MPICH-2 のインストールの手順を以下に列挙する。

1. MPICH-2 の公式ページ[5]より WINDOWS 用の MPI ソフト pich2-1.0.6p1-win32-ia32.msi をダウンロードする。
2. ダウンロードしたファイルを各計算機にインストールする。本研究では、各計算機のフォルダ”C:¥Program Files¥MPICH2”にインストールを行った。

3. MPICH-2 のバイナリのあるフォルダに対して各計算機の環境変数 PATH を指定する。本研究ではフォルダ”C:¥Program Files¥MPICH2¥bin”に対して環境変数 PATH の指定を行った。
4. 各計算機にネットワークを通して共有できるフォルダを設定する。本研究では、各計算機でフォルダ”C:¥mpi”を作成し、このフォルダのプロパティをネットワークを通じて共有できるように設定を行った。
5. 各計算機に MPICH-2 が使用するためのユーザを設定する。本研究では、各計算機に管理者権限を持つユーザ”mpi”を作成し、また、そのパスワードの設定を行った。

また、本研究では、プログラム言語として C/C++を用いた。C/C++のコンパイラは、VisualC++2008ExpressEdition が、マイクロソフトの公式ページ[11]より配布されているので、その仮想 CD をダウンロードしインストールを行うことができる。

MPICH-2 はライブラリが用意されているので、VisualC++のツールオプションから MPICH-2 のインクルードファイルおよびライブラリファイルのあるフォルダ”C:¥Program Files¥MPICH2¥include”および”C:¥Program Files¥MPICH2¥lib”を追加し、リンカ入力の依存ファイル”mpi.lib”を追加することにより、MPICH-2 を用いて並列プログラムを作成する

環境を作ることができる。

2.2.3 wav ファイル

wav[10]とは、Windows 標準の音声ファイルの形式である。「WAVE 形式」などとも呼ばれる事があり、音声信号をデジタルデータに変換したものを記録するための保存形式などを規定している。圧縮方式については規定しておらず、任意のものを利用することができる。

本研究で使用した wav ファイルの詳細を表３に示す。

表３　wav ファイル

サイズ	100MB	150MB	300MB	600MB
ビットレート	1411 kbps	1411 kbps	1411 kbps	1411 kbps
オーディオ サンプル サイズ	16ビット	16ビット	16ビット	16ビット
チャンネル	2(ステレオ)	2(ステレオ)	2(ステレオ)	2(ステレオ)
オーディオ サンプル レート	44KHz	44KHz	44KHz	44KHz
オーディオ形式	PCM	PCM	PCM	PCM

2.2.4 mp3 ファイル

mp3[9][10]とは、映像データ圧縮方式の MPEG-1 で利用される最も広く普及している音声圧縮方式の１つ。他の主要な音声圧縮方式と同様に、人間の感じ取りにくい部分のデータを間引くことによって高い圧縮率を得る非可逆圧縮方式を採用している。

本研究で変換処理を行い圧縮された mp3 ファイルの詳細を表４に示す。

表４　mp3 ファイル

wavファイル時のサイズ	100MB	150MB	300MB	600MB
変換後のサイズ	9.07MB(9516930バイト)	13.6MB(14269126バイト)	27.1MB(28511921バイト)	54.3MB(56971598バイト)
ビットレート	128kbps	128kbps	128kbps	128kbps
チャンネル	2(ステレオ)	2(ステレオ)	2(ステレオ)	2(ステレオ)
オーディオ サンプル レート	44KHz	44KHz	44KHz	44KHz

3　mp3 の並列エンコード

3.1 mp3 のエンコード

本研究では、MPI を用いて複数の wav ファイルを mp3 にエンコードし以下の検証を行う。

図１に本研究で使用した計算機ネットワークの概念図を示す。

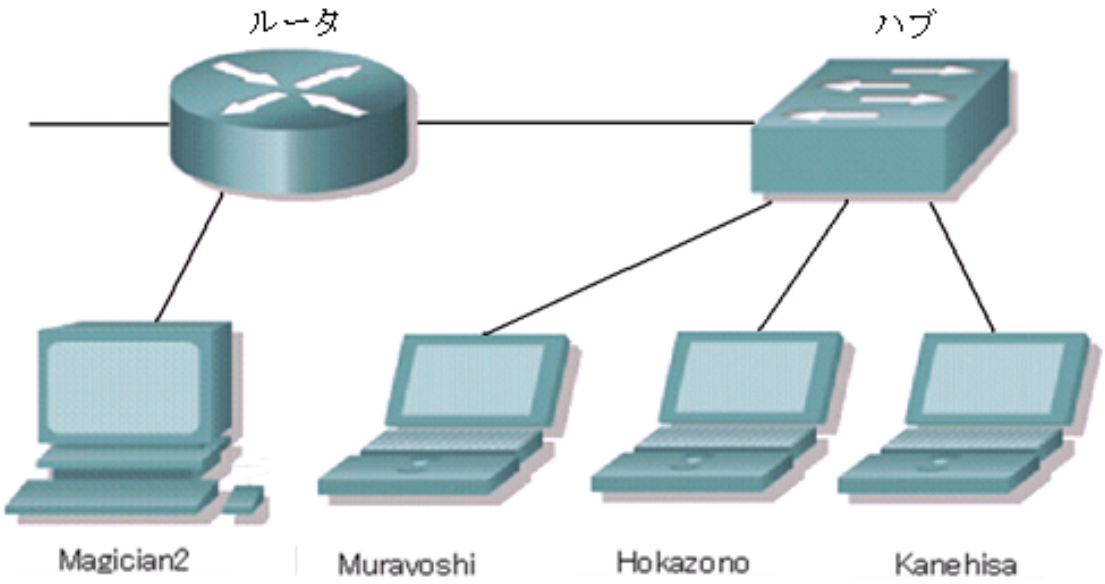

図１計算機ネットワークの概念図

① 性能の異なった４台の PC(Murayoshi,Hokazono,Kanehisa,Magician2)に性能の高いものから順に、予め分割しておいた wav ファイルをサイズの大きなものから振り分け mp3 ファイルに変換し、別に用意した性能の高い１台の PC(FM)で分割する前のファイルの変換を行った場合との処理時間の違いを検証する。振り分けたファイルは表１に記載する。

② ４台の PC に均等のサイズのファイルを振り分け変換を行った場合と①で行ったスペック毎にファイルサイズを変更して分割を行った場合の処理時間の違いを検証する。

使用するファイルはスペック毎に振り分ける為に 100MB と 300MB に、均等に振り分ける為に 150MB に予め分割してある。誤差を無くす為に全ての処理を 10 回ずつ行いその平均の数値を採用している。

3.2 mp3 エンコーダ

本研究で使用したエンコーダは、lame と呼ばれるエンコーダを使用しているダイナミックリングライブラリ gogo.dll[8]を使用した。gogo.dll は wav から mp3 へのエンコーダを簡略化して提供している。

以下に gogo.dll を用いての mp3 へのエンコード手順について説明する。

1. gogo.dll をメモリに読み込む。

2. ワークエリアの初期関数を呼び出す。
3. エンコード条件を設定する。
4. 条件の確定関数を呼び出す。
5. (必要であれば)確定した条件を取得する。
6. １フレームのエンコード関数を繰り返して呼び出す。
7. エンコード終了の関数を呼び出す。
8. gogo.dll の終了処理関数を呼び出す。
9. gogo.dll の開放をする。

複数のファイルをエンコードする場合、2.～8.を繰り返し呼び出す。

3.3 並列エンコード

本章では、本研究で作成した並列エンコーダの実行手順について述べる。
まず、実験前に以下の準備を行う。

1. 本研究で作成した並列エンコーダの実行ファイルを各 PC の”C:¥mpi”フォルダに置く。
2. 入力となる wav ファイルを各 PC の”C:¥mpi”フォルダに置く。ただし、wav ファイルのファイル名はそれぞれ”audio_1.wav”,”audio_2.wav”…とする。

MPICH は、実行時に各プロセスにランクが自動的に割り当てられる。そこで、各プロセスへの wav ファイルの振り分けはこのランクにより行うことができる。つまり、ランク n のプロセスに対しては ”audio_n.wav ” を割り当てれば良い。ここで注意しておくことは、wav ファイルの数以上にランクを指定してエンコードすることはできないことである。例えば、エンコードする wav ファイルが 8 個しか無いのに MPICH でプロセス数を 10 にすることはできない。ランク 9,10 を持つプロセスはそれぞれ ”C:¥mpi¥audio_9.wav ”および ”C:¥mpi¥audio_10.wav ”, を開こうとするのでエラーが起きるからである。計算機台数は何台あっても問題無いが、エンコードするファイルの数をプロセス数と同じにしなければエラーが発生する。
各計算機は割り当てられた wav ファイルを mp3 に変換し、実行後はそれぞれのホストに変換された mp3 ファイルが保存される。

3.4 フォルダの共有

本研究では並列エンコーダを実行する為にフォルダの共有を行う必要がある。以下にこの手順を示す。本研究では ” C:¥mpi ” を共有ファイルとしている。

1. ” C:¥mpi ” を右クリックし、「共有とセキュリティ」をクリックする。
2. 「共有のプロパティ」ダイアログが表示されるので、「ネットワーク上でこのフォルダを共有する」と「ネットワークユーザーによるファイルの変更を許可する」にチェックを入れる。
3. 「共有名」を入力する。ここでは共有名を ” mpi ” とする。最後に「OK」を押して終了

である。

4　結果・考察

本研究では、wav ファイルのサイズを PC のスペック毎に振り分けた場合と均等に振り分けた場合で mp3 に変換しその処理時間の計測を行った。表４に実験の計測結果を示す。処理時間の決定は誤差の無いように全ての処理を 10 回ずつ行いその平均の数値を採用している。

表４　計測結果（秒)

		PC名				
ファイルサイズ	ホスト数	Murayoshi	Hokazono	Kanehisa	Magichan2	FM
600MB	1	197.16	85.87	102.42	53.94	51.89
100MBと300MB	4	18.12				
150MB	4	20.02				

表４より、PC１台で処理を行うよりも４台で行った方が処理時間が短くなっていることがわかる。そして、wav ファイルを均等に PC に振り分けた場合よりもスペック毎にファイルサイズを変更して変換を行った方が少しではあるが時間の短縮が行えている事がわかる。

従って、wav ファイルから mp3 ファイルへのエンコードは、計算機台数の増加、ならびにスペック毎のファイルの振り分けにより効率よく実行時間の短縮が得られたことが示される。

5　結論

本研究では、MPICH2 による仮想並列環境の下で wav 形式のファイルから mp3 形式のファイルへのエンコードを行い、その実行結果を測定することで、どの程度の時間短縮が行えるのかを検証した。

本研究の計測結果により、wav ファイルから mp3 ファイルへのエンコードでは、高スペックの PC と低スペックの PC でそれぞれ処理するファイルのサイズを変更した方が処理時間の短縮に繋がる事がわかった。そして、低スペックの PC でも複数台で並列処理を行えば高スペックの PC１台の処理速度より速くなる事がわかった。

本研究ではファイルを分割し、それぞれの PC に適したサイズのファイルを振り分ける動作と、処理後に圧縮されたファイルを結合する動作の処理時間を含めていない為、この点を考慮して実

験を行えばより正確な処理時間を出すことができる。この点が今後の課題である。

謝辞

本研究を行うにあたって、お世話になった全ての人達に感謝の意を表したい。その中でも、並列処理について基礎から教えて下さった石水助教と同じ研究を行うにあたり常に励まし続けてくれた情報論理工学の研究室のメンバーには心から感謝しています。

参考文献

[1] P.パチェコ 著. 秋葉博 訳：MPI 並列プログラミング.培風館(2001)

[2] W.グロップ.E.ラスク.T.タークル著. 畑崎隆雄 訳：実践 MPI-2 メッセージパッシング・インターフェースの上級者向け機能, ピアソン・エデュケーション(2002)

[3] 渡邊真也 著：MPI による並列プログラミングの基礎, http://mikilab.doshisha.ac.jp/dia/smpp/cluster2000/PDF/chapter02.pdf

[4] Argonne National Laboratory, http://www.mcs.anl.gov/research/projects/mpich2/indexold.html

[5] MPICH2, http://www.mcs.anl.gov/research/projects/mpich2/

[6] PVM, Parallel Virtual Machine, http://www.csm.ornl.gov/pvm/

[7] PVM, http://erpc1.naruto-u.ac.jp/~geant4/pvm/pvm.html

[8] 午後のこ〜だ オンラインマニュアル, http://www.marinecat.net/free/windows/gogohelp/

[9] 大澤文孝：たちまちわかる MP3. 工学社(1999)

[10] 第一 I/O 編集部 編：音声・動画・文書ファイルの形式の達人になる本, 工学社(2002)

[11] Visual Studio 2008 Express Editions, http://www.microsoft.com/japan/msdn/vstudio/express/default.aspx

付録

本研究で用いた並列エンコーダのプログラムを以下に示す。

1.encoder.cpp

2.stab.cpp

3.musenc.h

また gogo.dll のソースコードを http://www.marinecat.net/free/windows/mct_free.htm からダウンロードし、コンパイルして gogo.dll を用意しておく必要がある。

付録.1.encoder.cpp

```
/*
 *  コンパイル時stab.cppを一緒にコンパイルしてください
 */
#define MPICH_SKIP_MPICXX
#include "mpi.h"
#include <stdio.h>
#include <windows.h>
#include "musenc.h"
#include <time.h>
#include <stdlib.h>

/*
ファイル名と拡張子を分けて後で結合する
audio_の後には任意の数字が結合され.wavが次に結合される
-->C:¥mpi¥audio_1.wav
*/
#define FILE "C:¥¥mpi¥¥audio_"

int ErrorCheck(MERET rval){
        switch(rval){
                case ME_NOERR:return 1;break;
                case ME_EMPTYSTREAM:return 1;break;
                case ME_HALTED:printf("中断されました¥n");return -1;break;
                case ME_INTERNALERROR:printf("内部エラーが発生しました¥n");return -1;break;
                case ME_PARAMERROR:printf("設定パラメーターのエラー¥n");return -1;break;
                case ME_NOFPU:printf("x87FPUを装着していません¥n");return -1;break;
                case ME_INFILE_NOFOUND:printf("入力ファイルを正しく開けません¥n");return -1;break;
```

```
                case ME_OUTFILE_NOFOUND:printf("出力ファイルを正しく開けません¥n");return -1;break;
                case ME_FREQERROR:printf("入出力周波数が正しくありません¥n");return -1;break;
                case ME_BITRATEERROR:printf("出力ビットレートが正しくありません¥n");return -1;break;
                case ME_WAVETYPE_ERR:printf("ウェーブタイプが正しくありません¥n");return -1;break;
                case ME_CANNOT_SEEK:printf("正しくシーク出来ません¥n");return -1;break;
                case ME_BITRATE_ERR:printf("ビットレート設定が正しくありません¥n");return -1;break;
                case ME_BADMODEORLAYER:printf("モードの設定が正しくありません¥n");return -1;break;
                case ME_NOMEMORY:printf("メモリアローケーションに失敗しました¥n");return -1;break;
                case ME_CANNOT_CREATE_THREAD:printf("スレッド生成エラー¥n");return -1;break;
                case ME_WRITEERROR:printf("記憶媒体の容量不足です¥n");return -1;break;
                default:return -1;
        }
}

//フレーム単位でエンコードする
MERET frame_encoder(MERET rval,UPARAM totalFrame,UPARAM curFrame)
{
        do {
                //printf("%d / %d (%d%%)¥r", curFrame,
                //      totalFrame,curFrame / ((totalFrame + 99)/100) );
                curFrame++;
                // 1フレームエンコードを繰り返す
                rval = MPGE_processFrame();
                // 入力ストリームがなくなる(ME_EMPTYSTREAM) or
                // その他エラーが発生するまで繰り返し。
                } while(rval == ME_NOERR);

                return rval;
}

int main(int argc, char **argv)
{
        MPI_Comm mpi_comm;
        //MPI_Status mpi_stat;
        int num_proc,myrank,proc_name_len;
        char proc_name[10];

        static char filename[256];//="file" + "(myrank+1)" + "extension"
        char file[] = FILE;//audio_X.mp3で出力する(Xは任意の数字)
```

```
char extension[]=".wav";//拡張子.wav。filenameに結合するためのファイル
MERET    rval;
double ts,te,tp;//時間測定のため


MPI_Init(&argc,&argv);  //MPIライブラリを使用するための準備(初期化)を行う
mpi_comm = MPI_COMM_WORLD;

MPI_Comm_size(mpi_comm, &num_proc);

MPI_Comm_rank(mpi_comm, &myrank);
MPI_Get_processor_name(proc_name, &proc_name_len);
MPI_Barrier(mpi_comm);
ts=MPI_Wtime();


/*****************/
//MPI振り分け処理
/*****************/
if(myrank==0){
        printf("%s is rank:%d 処理中¥n",proc_name,myrank);
        // 1. DLL読み込み&初期化
        rval = MPGE_initializeWork();
        if(!ErrorCheck(rval))return -1;

        // 2. ファイル名の設定
        sprintf_s(filename,"%s%d%s",file,1,extension);

        rval=MPGE_setConfigure( MC_INPUTFILE, MC_INPDEV_FILE, (UPARAM)filename);
        if(!ErrorCheck(rval))return -1;

        // 3. パラメータ解析
        rval = MPGE_detectConfigure();
        if(!ErrorCheck(rval))return -1;

        // 全フレーム数を取得
        UPARAM  totalFrame, curFrame;
        MPGE_getConfigure( MG_COUNT_FRAME, (UPARAM*)&totalFrame);
        curFrame = 0;
```

```
        //エンコード
        rval = frame_encoder(rval,totalFrame,curFrame);

        //エンコードが終わってストリームが最後まで達したかどうか
        ErrorCheck(rval);

        // 5.エンコーダーを閉じる
        MPGE_closeCoder();
        printf("%s is rank %d:  %s -> %s%d.mp3¥n",proc_name,myrank,filename,file,myrank+1);
}
else{
        printf("%s is rank:%d 処理中¥n",proc_name,myrank);
        // 1. DLL読み込み&初期化
        rval = MPGE_initializeWork();
        if(!ErrorCheck(rval))return -1;

        // 2. ファイル名の設定
        sprintf_s(filename,"%s%d%s",file,1+myrank,extension);

        rval=MPGE_setConfigure( MC_INPUTFILE, MC_INPDEV_FILE, (UPARAM)filename);
        if(!ErrorCheck(rval))return -1;

        // 3. パラメータ解析
        rval = MPGE_detectConfigure();
        if(!ErrorCheck(rval))return -1;

        // 全フレーム数を取得
        UPARAM  totalFrame, curFrame;
        MPGE_getConfigure( MG_COUNT_FRAME, (UPARAM*)&totalFrame);
        curFrame = 0;

        //エンコード
        rval = frame_encoder(rval,totalFrame,curFrame);

        //エンコードが終わってストリームが最後まで達したかどうか
        ErrorCheck(rval);

        // 5.エンコーダーを閉じる
```

```
                MPGE_closeCoder();
                printf("%s is rank %d:  %s -> %s%d.mp3¥n",proc_name,myrank,filename,file,myrank+1);
        }

        MPI_Barrier(mpi_comm);
        te=MPI_Wtime();
        tp=MPI_Wtick();

        if(myrank == 0){
                printf("Process time:%lf¥n",te-ts);
                printf("Precision:%lf¥n", tp);
        }


        // 6.DLL終了& 開放
        MPGE_endCoder();

        MPI_Finalize();

        return 0;
}
```

付録.2.stab.cpp

```
#include <windows.h>
#include <windowsx.h>
#include <winuser.h>
#include <stdio.h>
//#include "resource.h"
#include "musenc.h"

static  HINSTANCE       hModule = NULL;
typedef MERET (*me_init)(void);
typedef MERET (*me_setconf)(MPARAM mode, UPARAM dwPara1, UPARAM dwPara2 );
typedef MERET (*me_getconf)(MPARAM mode, void *para1 );
typedef MERET (*me_detect)();
typedef MERET (*me_procframe)();
typedef MERET (*me_close)();
typedef MERET (*me_end)();
typedef MERET (*me_getver)( unsigned long *vercode,  char *verstring );
```

```
typedef MERET (*me_haveunit)( unsigned long *unit );

static  me_init         mpge_init;
static  me_setconf      mpge_setconf;
static  me_getconf      mpge_getconf;
static  me_detect       mpge_detector;
static  me_procframe mpge_processframe;
static  me_close mpge_close;
static  me_end          mpge_end;
static  me_getver       mpge_getver;
static  me_haveunit mpge_haveunit;

// DLLの読み込み(最初の回目のみ)とワークエリアの初期化を行います。
MERET   MPGE_initializeWork()
{
        if( hModule == NULL ){
                // (DLLが読み込まれていない場合)
                // カレントディレクトリ、及びsystemディレクトリのGOGO.DLLの読み込み
                hModule = LoadLibrary("gogo.dll");
                if( hModule == NULL ){                  // DLLが見つからない場合
                        #define Key             HKEY_CURRENT_USER
                        #define SubKey "Software¥¥MarineCat¥¥GOGO_DLL"
                        HKEY    hKey;
                        DWORD   dwType, dwKeySize;
                        LONG    lResult;
                        static  char    *szName = "INSTPATH";
                        char    szPathName[ _MAX_PATH + 8];
                        dwKeySize = sizeof( szPathName );

                        // レジストリ項目のHEY_CURENT_USER¥Software¥MarineCat¥GOGO_DLLキー以下の
                        // INSTPATH (REG_SZ)を取得します。
                        if( RegOpenKeyEx(
                                        Key,
                                        SubKey,
                                        0,
                                        KEY_ALL_ACCESS,
                                        &hKey ) == ERROR_SUCCESS
                        ){
                                lResult = RegQueryValueEx(
```

```
                                        hKey,
                                        szName,
                                        0,
                                        &dwType,
                                        (BYTE *)szPathName,
                                        &dwKeySize);
                                RegCloseKey(hKey);
                                if( lResult == ERROR_SUCCESS && REG_SZ == dwType ){
                                        // レジストリから取得したパスで再度DLLの読み込みを試みる
                                        hModule = LoadLibrary( szPathName );
                                }
                        }
                }
                // DLLが見つからない
                if( hModule == NULL ){
//                      MessageBox( "DLLの読み込みを失敗しました。¥nDLLをEXEファイルと同じディレクトリへ複写してください¥n");
                        fprintf( stderr,"DLLの読み込みを失敗しました。¥nDLLをEXEファイルと同じディレクトリへ複写してください¥n");
                        exit( -1 );
                }

                // エクスポート関数の取得
                mpge_init = (me_init )GetProcAddress( hModule, "MPGE_initializeWork" );
                mpge_setconf = (me_setconf )GetProcAddress( hModule, "MPGE_setConfigure" );
                mpge_getconf = (me_getconf )GetProcAddress( hModule, "MPGE_getConfigure" );
                mpge_detector = (me_detect )GetProcAddress( hModule, "MPGE_detectConfigure" );
                mpge_processframe = (me_procframe )GetProcAddress( hModule, "MPGE_processFrame" );
                mpge_close   = (me_close )GetProcAddress( hModule, "MPGE_closeCoder" );
                mpge_end = (me_end )GetProcAddress( hModule, "MPGE_endCoder" );
                mpge_getver       = (me_getver )GetProcAddress( hModule, "MPGE_getVersion" );
                mpge_haveunit= (me_haveunit )GetProcAddress( hModule, "MPGE_getUnitStates" );
        }

        // すべての関数が正常か確認する
        if( mpge_init && mpge_setconf && mpge_getconf &&
                mpge_detector && mpge_processframe && mpge_end && mpge_getver && mpge_haveunit )
                return (mpge_init)();
```

```
        // エラー
        fprintf( stderr, "DLLの内容を正しく識別することが出来ませんでした¥n");
        FreeLibrary( hModule );
        hModule = NULL;
        exit( -1 );

        return ME_NOERR;
}

MERET   MPGE_setConfigure(MPARAM mode, UPARAM dwPara1, UPARAM dwPara2 )
{
        return (mpge_setconf)( mode, dwPara1, dwPara2 );
}

MERET   MPGE_getConfigure(MPARAM mode, void *para1 )
{
        return (mpge_getconf)( mode, para1 );
}

MERET   MPGE_detectConfigure()
{
        return (mpge_detector)();
}

MERET   MPGE_processFrame()
{
        return (mpge_processframe)();
}

MERET   MPGE_closeCoder()
{
        return (mpge_close)();
}

MERET   MPGE_endCoder()
{
        MERET    val = (mpge_end)();
        if( val == ME_NOERR ){
                FreeLibrary( hModule );          // DLL開放
```

```
                hModule = NULL;
        }
        return val;
}

MERET   MPGE_getVersion( unsigned long *vercode,  char *verstring )
{
        return (mpge_getver)( vercode, verstring );
}

MERET   MPGE_getUnitStates( unsigned long *unit)
{
        return   (mpge_haveunit)( unit );
}
```

付録.3.musenc.h

```
#ifndef __MUSUI_H__
#define __MUSUI_H__

#include <limits.h>

typedef signed int                              MERET;
#ifndef __os2__
typedef unsigned long                   MPARAM;
#else
typedef  unsigned long          MUPARAM;
#endif
typedef unsigned long                   UPARAM;

#ifdef GOGO_DLL_EXPORTS
#define         EXPORT                          __declspec(dllexport)
#else
#define         EXPORT
#endif


#define ME_NOERR                                (0)             // return normally;正常終了
#define ME_EMPTYSTREAM                  (1)             // stream becomes empty;ストリーム
```

```
が最後に達した
#define ME_HALTED                                   (2)             // stopped by user;(ユーザ
ーの手により)中断された
#define ME_INTERNALERROR                (10)    // internal error; 内部エラー
#define ME_PARAMERROR                   (11)    // parameters error;設定でパラメーターエラー
#define ME_NOFPU                        (12)    // no FPU;FPUを装着していない!!
#define ME_INFILE_NOFOUND               (13)    // can't open input file;入力ファイルを正し
く開けない
#define ME_OUTFILE_NOFOUND              (14)    // can't open output file;出力ファイルを正し
く開けない
#define ME_FREQERROR                    (15)    // frequency is not good;入出力周波数が正し
くない
#define ME_BITRATEERROR                 (16)    // bitrate is not good;出力ビットレートが正
しくない
#define ME_WAVETYPE_ERR                 (17)    // WAV format is not good;ウェーブタイプが正
しくない
#define ME_CANNOT_SEEK                  (18)    // can't seek;正しくシーク出来ない
#define ME_BITRATE_ERR                  (19)    // only for compatibility;ビットレート設定が
正しくない
#define ME_BADMODEORLAYER               (20)    // mode/layer not good;モード・レイヤの設定
異常
#define ME_NOMEMORY                                 (21)    // fail to allocate memory;メモリア
ローケーション失敗
#define ME_CANNOT_SET_SCOPE             (22)    // thread error;スレッド属性エラー(pthread
only)
#define ME_CANNOT_CREATE_THREAD         (23)    // fail to create thear;スレッド生成エラー
#define ME_WRITEERROR                   (24)    // lock of capacity of disk;記憶媒体の容量不
足


// definition of call-back function for user;ユーザーのコールバック関数定義
typedef MERET    (*MPGE_USERFUNC)(void *buf, unsigned long nLength );
#define MPGE_NULL_FUNC (MPGE_USERFUNC)NULL          // for HighC

////////////////////////////////////////////////////////////////////////
// Configuration
////////////////////////////////////////////////////////////////////////
// for INPUT
#define         MC_INPUTFILE                    (1)
```

```
// para1 choice of input device
        #define         MC_INPDEV_FILE          (0)             // input device is file;入力デバイスはファイル
        #define         MC_INPDEV_STDIO         (1)             //                 stdin;入力デバイスは標準入力
        #define         MC_INPDEV_USERFUNC      (2)             //       defined by user;入力デバイスはユーザー定義
        // para2 (必要であれば)ファイル名。ポインタを指定する
        // メモリよりエンコードの時は以下の構造体のポインタを指定する.
        struct MCP_INPDEV_USERFUNC {
                MPGE_USERFUNC   pUserFunc;                      // pointer to user-function for call-back or MPGE_NULL_FUNC if none
                                                                                                // コールバック対象のユーザー関数。未定義時MPGE_NULL_FUNCを代入
                unsigned int    nSize;                          // size of file or MC_INPDEV_MEMORY_NOSIZE if unknown
                                                                                                // ファイルサイズ。不定の時はMC_INPDEV_MEMORY_NOSIZEを指定
                int                             nBit;                           // nBit = 8 or 16 ; PCMビット深度を指定
                int                             nFreq;                          // input frequency ; 入力周波数の指定
                int                             nChn;                           // number of channel(1 or 2) ; チャネル数
        };
        #define         MC_INPDEV_MEMORY_NOSIZE         (UINT_MAX)
/*
  Using userfunction input;
  ユーザー関数利用時の挙動
  ^^^^^^^^^^^^^^^^^^^^^^^^


  ユーザーが登録した関数UsefFuncに対して、DLLより読み込み要求が行われる。
  MERET  UserFunc_input(void *buf, unsigned long nLength );

  要求を処理する際に
     ・void *buf にはnLength バイト分のデータを格納、return ME_NOERRで抜ける
        ・ファイルの最後に達して、nLength分読み込めない(かつ少なくともバイト以上読み込める)場合、
          memset( buf + 読み込んだデータbyte, 0, nLength - 読み込んだデータサイズ) ;
         としてreturn ME_NOERR する。
```

```
        ・１バイトも読めない場合は、何もせずreturn ME_EMPTYSTREAM; で抜ける
*/

//////////////////////////////////////////////////////////////////////
// for OUTPUT ( now stdout is not support )
#define          MC_OUTPUTFILE                    (2)
// para1 choice of output device
        #define          MC_OUTDEV_FILE           (0)             // output device is file;出力デバイスはファイル
        #define          MC_OUTDEV_STDOUT (1)             //                 stdout; 出力デバイスは標準出力
        #define          MC_OUTDEV_USERFUNC       (2)             //       defined by user;出力デバイスはユーザー定義
        #define          MC_OUTDEV_USERFUNC_WITHVBRTAG    (3)     //       defined by user;入力デバイスはユーザー定義/VBRタグ書き出し
// para2 pointer to file if necessary ;(必要であれば)ファイル名。ポインタ指定

/*
  Using userfunction output
  ユーザー関数利用時の挙動
  ^^^^^^^^^^^^^^^^^^^^^^^^

  ユーザーが登録した関数UsefFuncに対して、DLLより書込み要求が行われる。
  MERET  UserFunc_output(void *buf, unsigned long nLength );

  要求を処理する際に
     ・void *buf にはnLength バイト分のデータが格納されているので
          fwrite( buf, 1, nLength, fp );の様にして書き出しreturn ME_NOERRで抜ける.
          書き出しに失敗した時は、return ME_WRITEERROR;で抜ける.
        ・最後にbuf == NULLで度呼び出される. return 値は何でも良い。
       (MC_OUTDEV_USERFUNC_WITHVBRTAGで登録した際には、以下の挙動が追加される)
        ・もう一度buf == NULLで呼び出される.この際にファイルの先頭へシークし、
          ファイル全体のサイズをreturnの値とする。filesize<=0の時は終了。
          (誤ってreturn ME_NOERR; で抜けない様に注意!! )
        ・XING-VBRタグデータがbufに、XINGVBRタグのサイズがnLengthに格納されて呼び出される.
        ・最後にもう一度buf == NULLで呼び出される.
*/

//////////////////////////////////////////////////////////////////////
```

```
// mode of encoding ;エンコードタイプ
#define          MC_ENCODEMODE                  (3)
// para1 mode;モード設定
        #define          MC_MODE_MONO           (0)          // mono;モノラル
        #define          MC_MODE_STEREO         (1)          // stereo;ステレオ
        #define          MC_MODE_JOINT          (2)          // joint-stereo;ジョイント
        #define          MC_MODE_MSSTEREO (3)          // mid/side stereo;ミッドサイド
        #define          MC_MODE_DUALCHANNEL    (4)          // dual channel;デュアルチャネル

//////////////////////////////////////////////////////////////////////////
// bitrate;ビットレート設定
#define          MC_BITRATE                             (4)
// para1 bitrate;ビットレート即値指定


//////////////////////////////////////////////////////////////////////////
// frequency of input file (force);入力で用いるサンプル周波数の強制指定
#define          MC_INPFREQ                             (5)
// para1 frequency;入出力で用いるデータ

//////////////////////////////////////////////////////////////////////////
// frequency of output mp3 (force);出力で用いるサンプル周波数の強制指定
#define          MC_OUTFREQ                             (6)
// para1 frequency;入出力で用いるデータ

//////////////////////////////////////////////////////////////////////////
// size ofheader if you ignore WAV-header (for example cda);エンコード開始位置の強制指定(ヘッダを無視する時)
#define          MC_STARTOFFSET                 (7)

//////////////////////////////////////////////////////////////////////////
// psycho-acoustics ON/OFF;心理解析ON/OFF
#define          MC_USEPSY                              (8)
// PARA1 boolean(TRUE/FALSE)

//////////////////////////////////////////////////////////////////////////
// 16kHz low-pass filter ON/OFF;16KHz低帯域フィルタON/OFF
#define          MC_USELPF16                            (9)
// PARA1 boolean(TRUE/FALSE)
```

```
////////////////////////////////////////////////////////////////////////
// use special UNIT, para1:boolean; ユニット指定para1:BOOL値
#define          MC_USEMMX                              (10)     // MMX
#define          MC_USE3DNOW                            (11)     // 3DNow!
#define          MC_USEKNI                              (12)     // SSE(KNI)
#define          MC_USEE3DNOW                  (13)     // Enhanced 3D Now!
#define          MC_USESPC1                             (14)     // special switch for debug
#define          MC_USESPC2                             (15)     // special switch for debug

////////////////////////////////////////////////////////////////////////
// addition of TAG; ファイルタグ情報付加
#define          MC_ADDTAG                              (16)
// dwPara1  length of TAG;タグ長
// dwPara2  pointer to TAG;タグデータのポインタ

////////////////////////////////////////////////////////////////////////
// emphasis;エンファシスタイプの設定
#define          MC_EMPHASIS                            (17)
// para1 type of emphasis;エンファシスタイプの設定
        #define          MC_EMP_NONE                    (0)              // no empahsis;エンファシス
なし(dflt)
        #define          MC_EMP_5015MS          (1)              // 50/15ms    ;エンファシス/15ms
        #define          MC_EMP_CCITT           (3)              // CCITT      ;エンファシスCCITT

////////////////////////////////////////////////////////////////////////
// use VBR;VBRタイプの設定
#define          MC_VBR                                 (18)

////////////////////////////////////////////////////////////////////////
// SMP support para1: interger
#define          MC_CPU                                 (19)

////////////////////////////////////////////////////////////////////////
// for RAW-PCM; 以下つはRAW-PCMの設定のため
// byte swapping for 16bitPCM; PCM入力時のlow, high bit 変換
#define          MC_BYTE_SWAP                  (20)

////////////////////////////////////////////////////////////////////////
```

```
// for 8bit PCM
#define          MC_8BIT_PCM                                (21)

////////////////////////////////////////////////////////////////////////
// for mono PCM
#define          MC_MONO_PCM                                (22)

////////////////////////////////////////////////////////////////////////
// for Towns SND
#define          MC_TOWNS_SND                       (23)

////////////////////////////////////////////////////////////////////////
// BeOS & Win32 Encode thread priority
#define          MC_THREAD_PRIORITY                 (24)
// (WIN32) dwPara1 MULTITHREAD Priority (THREAD_PRIORITY_**** at WinBASE.h )

////////////////////////////////////////////////////////////////////////
// BeOS Read thread priority
//#if    defined(USE_BTHREAD)
#define          MC_READTHREAD_PRIORITY    (25)
//#endif

////////////////////////////////////////////////////////////////////////
// output format
#define          MC_OUTPUT_FORMAT          (26)
// para1
        #define          MC_OUTPUT_NORMAL (0)              // mp3+TAG(see MC_ADDTAG)
        #define          MC_OUTPUT_RIFF_WAVE       (1)             // RIFF/WAVE
        #define          MC_OUTPUT_RIFF_RMP        (2)             // RIFF/RMP

////////////////////////////////////////////////////////////////////////
// LIST/INFO chunk of RIFF/WAVE or RIFF/RMP
#define          MC_RIFF_INFO                       (27)
// para1 size of info(include info name)
// para2 pointer to info
//   offset            contents
//   0..3              info name
//   4..size of info-1 info
```

```
////////////////////////////////////////////////////////////////////////
// verify and overwrite
#define          MC_VERIFY                      (28)

////////////////////////////////////////////////////////////////////////
// output directory
#define          MC_OUTPUTDIR                   (29)

////////////////////////////////////////////////////////////////////////
// VBRの最低/最高ビットレートの設定
#define          MC_VBRBITRATE                  (30)
// para1 最低ビットレート(kbps)
// para2 最高ビットレート(kbps)


////////////////////////////////////////////////////////////////////////
// 拡張フィルタの使用LPF1, LPF2
#define          MC_ENHANCEDFILTER              (31)
// para1 LPF1 (0-100)
// para2 LPF2 (0-100)


////////////////////////////////////////////////////////////////////////
// Joint-stereoにおける、ステレオ/MSステレオの切り替えの閾値
#define          MC_MSTHRESHOLD                 (32)
// para1 threshold  (0-100)
// para2 mspower    (0-100)

////////////////////////////////////////////////////////////////////////
// Language
#define          MC_LANG                        (33)
// t_lang defined in message.h

MERET    EXPORT   MPGE_initializeWork();
#ifndef __os2__
MERET    EXPORT   MPGE_setConfigure(MPARAM mode, UPARAM dwPara1, UPARAM dwPara2 );
MERET    EXPORT   MPGE_getConfigure(MPARAM mode, void *para1 );
#else
MERET EXPORT   MPGE_setConfigure(MUPARAM mode, UPARAM dwPara1, UPARAM dwPara2 );
```

```
MERET EXPORT   MPGE_getConfigure(MUPARAM mode, void *para1 );
#endif
MERET    EXPORT   MPGE_detectConfigure();
#ifdef USE_BETHREAD
MERET    EXPORT   MPGE_processFrame(int *frameNum);
#else
MERET    EXPORT   MPGE_processFrame();
#endif
MERET    EXPORT   MPGE_closeCoder();
MERET    EXPORT   MPGE_endCoder();
MERET    EXPORT   MPGE_getUnitStates( unsigned long *unit );
MERET    EXPORT   MPGE_processTrack(int *frameNum);

// This function is effective for gogo.dll;このファンクションはDLLバージョンのみ有効
MERET    EXPORT   MPGE_getVersion( unsigned long *vercode,  char *verstring );
// vercode = 0x125 ->  version 1.25
// verstring       ->  "ver 1.25 1999/09/25" (allocate abobe 260bytes buffer)



////////////////////////////////////////////////////////////////////////////
// for getting configuration
////////////////////////////////////////////////////////////////////////////

#define          MG_INPUTFILE                    (1)             // name of input file ;入力ファイル
名取得
#define          MG_OUTPUTFILE                   (2)             // name of output file;出力ファイル
名取得
#define          MG_ENCODEMODE                   (3)             // type of encoding   ;エンコードモ
ード
#define          MG_BITRATE                              (4)             // bitrate            ;ビ
ットレート
#define          MG_INPFREQ                              (5)             // input frequency    ;入力
周波数
#define          MG_OUTFREQ                              (6)             // output frequency   ;出力
周波数
#define          MG_STARTOFFSET                  (7)             // offset of input PCM;スタートオフ
セット
#define          MG_USEPSY                               (8)             // psycho-acoustics   ;心
```

```
理解析を使用する/しない
#define         MG_USEMMX                       (9)             // MMX
#define         MG_USE3DNOW                     (10)    // 3DNow!
#define         MG_USEKNI                       (11)    // SSE(KNI)
#define         MG_USEE3DNOW            (12)    // Enhanced 3DNow!

#define         MG_USESPC1                      (13)    // special switch for debug
#define         MG_USESPC2                      (14)    // special switch for debug
#define         MG_COUNT_FRAME          (15)    // amount of frame
#define         MG_NUM_OF_SAMPLES       (16)    // number of sample for 1 frame;1フレームあ
たりのサンプル数
#define         MG_MPEG_VERSION         (17)    // MPEG VERSION
#define         MG_READTHREAD_PRIORITY  (18)    // thread priority to read for BeOS

#endif /* __MUSUI_H__ */
```