卒業研究報告書

題目

MPI による音声の並列計算処理

指 導 教 員

石水 隆　助教

報告者

05-1-037-0200

河合章太

近畿大学理工学部情報学科

平成 22 年 2 月 5 日提出

概要

現在、様々な分野で計算処理の高速化が求められている。高速処理を行うためには、複数のプロセッサを持つ並列計算機(Parallel Computer)[17][18]が用いられる。しかし、一般に並列計算機は非常に高価であり、容易に用いることはできない。

そこで、複数の計算機をネットワーク接続して 1 台の仮想的な並列計算機とする仮想並列計算(Parallel Virtual Computing)が現在重視されている。仮想並列計算を導入すれば、ネットワークを用いて誰もが安価なコストで並列計算環境を得ることができ、ベンチマーク性能ではスーパーコンピュータに匹敵する処理速度を得ることが可能となる。

本研究では、無料提供されている、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)13[1][2][3]を用いて、wav(RIFF waveform Audio Format)[9]形式の音声データファイルを mp3(MPEG Audio Layer-3)[10]形式にエンコードしたときの実行時間を推測し、MPI の有用性を評価する。

目次

1. 序論

1.1 並列処理(Parallel Processing)と並列計算機(Parallel Computer)

並列処理(Parallel Processing)[16][18]は、複数のプロセッサなどに処理分散を分散して割り当て、システム全体の処理能力の向上を図る手法である。並列処理を用いることにより、処理時間の大幅な短縮が見込まれている。

一台のコンピュータにマイクロプロセッサを複数用意する「マルチプロセッサ」や、プロセッサ内部に複数の処理装置を実装する「マルチコア」など、主にコンピュータ内部で並列的に処理を行なう手法について用いられる語でもあるが、現在では、コンピュータシステム自体を並列に接続して処理を分散する技術なども含め、「並列コンピューティング」(パラレルコンピューティング)という語が用いられることが多い。
複数のプロセッサを用いることで並列処理を行なうことが可能な計算機を並列計算機(Parallel Computer)という。

1.2 並列処理方式

並列計算処理方法には、全プロセッサが単一の物理アドレス空間を共有し処理を行う、共有メモリ（shared memory）型と、各プロセッサがローカルな独自の物理アドレス空間を持ち処理を行う、分散メモリ（distributed memory）型の 2 つに大きく分けることが出来る。共有メモリ型並列計算機か事用の並列計算機が必要になるのに対し、分散型メモリ並列計算機は複数の並列計算機をネットワーク接続することにより仮想的に構築することができる。そこで本研究では並列処理方式として分散メモリ型を使用する。

1.3 仮想並列計算(Parallel Virtual Computing)

大規模なデータの高速処理には並列計算機が必要となるが、並列計算機自体はとても高価なものであり、並列計算機を持つのはごく一部の大学や研究所そして企業だったこともあり、注目はされてはいたが利用しにくい状況であった。このため、複数の計算機をネットワーク接続することにより 1 台の仮想的な並列計算機とする仮想並列計算(Parallel Virtual Computing)が現在注目されている。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、安価に並列計算環境を構築することが可能である。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]などが存在する。

1.4 PVM (Parallel Virtual Machine)

PVM(Parallel Virtual Machine)[6][7]は、1991 年に米国のオークリッジ国立研究所[4]を中心に開発された、メッセージパッシングによる並列処理を行なうための並列化ライブラリである。

PVM はワークステーションクラスタなため、TCP/IP の通信ライブラリで一般的に使用されている LAN 環境があれば並列処理が実行出来るので多くのユーザが利用している。また、異機種間の通信も考慮されているため、対応する計算機は家庭にあるパーソナルコンピュータからスーパーコンピュータなど多くの種類で PVM による並列処理が出来る。

PVM の構成は 2 つに大きく分けられる。1 つはデーモン(pdmd3)であり、PVM によって構成された仮想並列計算機上にある全ての計算機にデーモンが存在する。PVM はこのデーモンを使用し通信を行なっている。複数のユーザは互いにオーバーラップさせ仮想並列計算機を構成することが出来る、また、各ユーザは PVM アプリケーションを 1 人で複数実行することが可能となっている。

もう 1 つは、ルーチンライブラリ(libpvm3.a)である。このライブラリは、メッセージパッシング、プロセスの生成、タスクの協調などの必要な関数を仮想計算機の構成ルーチンを提供する。

PVM の特徴として、耐故障性があげられる。並列計算中に仮想並列計算機を構成する計算機のうち 1 台が停止してしまった場合、停止した計算機を仮想並列計算機内から迅速に削除されるようになっているため、耐故障性に優れている。また、異機種間でも動作が可能という利点もある。しかし、PVM が多くの並列計算機に移植されるようになったとき、各並列計算機ベンダが独自にチューニングした PVM 開発を行ったため、異なるベンダ間でのプログラムの移植性が乏しくなった。

1.5 MPI (Message Passing Interface)

MPI(Message Passing Interface)[1][2][3]は、1992 年に結成された MPI Forum(MPIF)により標準使用の定義や検討を作り始めたことで具体化してきた。MPI の開発には、アメリカ、ヨーロッパの 40 の組織から 60 人の人間が関わっており、研究者や主な並列計算機ベンダのほとんどが参加した。メッセージパッシングによる通信の仕組みを共有化することを目的に MPI の規格作成が開始され、1994 年に MPI のヴァージョン 1.0 がまとめられた。翌年 1995 年には新しい機能の拡張を考慮した MPI-2 が検討され、1997 年に規格がまとめられた。

MPI は標準を目指して作成されたために様々な通信関数が実装されている、MPI 規約を用いて作成したプログラムは移植性が高いため、MPI を使用するユーザは、通信を考慮せずプログラムを組むことが出来る。

大きく PVM と違う点は、異機種間の通信が考慮されていないことである、MPI を用いての仮想計算機の構築には使用する計算機のオペレーティングシステムが同じでなければならないという制約が存在する。

MPI は専用の並列計算機からワークステーション、パーソナルコンピュータに至るまで幅広くサポートされており、無料で提供されている主な実装は MPICH や LAM といったものがある。

MPI のサポートするプログラミング言語は多く、C 言語や Fortran そして最近では Java などに対応している。

MPI による仮想並列計算環境においての通信は TCP/IP などのネットワークを用いて行われる。仮想並列計算機を構成する各計算機はアーキテクチャにより通信方法が異なり、それに伴い実装も異なる。そのため、ユーザが通信方式の差異等を気にせずにすむように MPI では「MPI ライブラリ」が用意されている。

1.6　　本研究の目的

本研究では、ネットワーク接続された複数の計算機により構成する。MPI による仮想並列計算機の有用性を実験的に評価する。評価方法として wav(RIFF waveform Audio Format)[9]形式の音声データファイルを mp3(MPEG Audio Layer-3)[10]形式にエンコードしたときの実行時間を測定し、計算機 1 台でのエンコードの処理時間が、複数台の並列処理を用いたエンコード時間と比べ、どの程度時間短縮が可能となるかを検証する。

2.　研究内容

2.1　使用機器

表本研究では Windows 環境で MPI を構築する。本研究で使用する計算機の詳細を表 1、使用するネットワーク構成を図 1 に示す。

表 1　使用する計算機の詳細

計算機名	Kawai	Kuwaki	Mori	Ishida
OS	Windows XP Professional	Windows XP Professional	Windows XP Professional	Windows XP Professional
CPU	Pentium(R)M 1.5GHz	Pentium(R)M 1.5GHz	Pentium(R)M 1.5GHz	Pentium(R)M 1.5GHz
メモリ	512MB	512MB	512MB	512MB

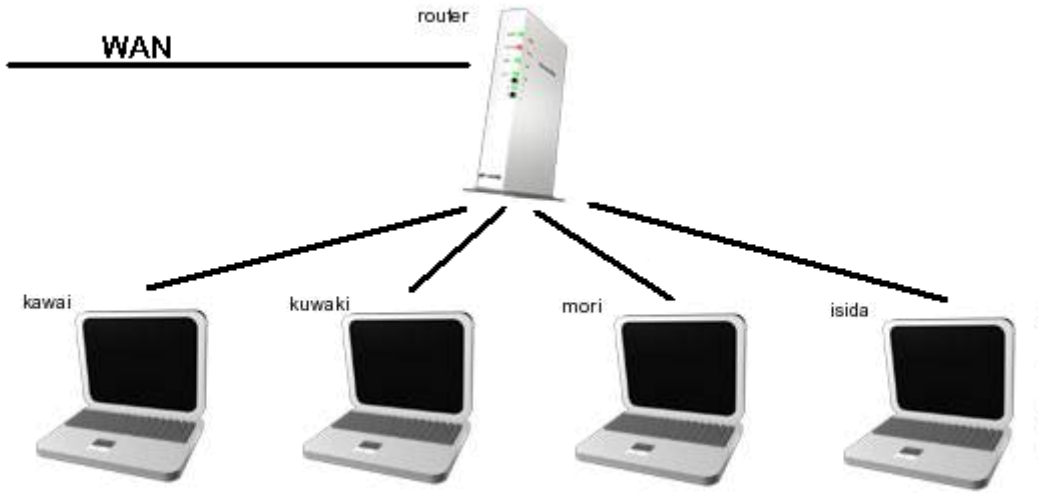

図 1 使用するネットワーク構成

2.2 使用音源

本研究ではMPICH2を用いて、wav(RIFF waveform Audio Format)[9]形式の音声ファイルをmp3(MPEG Audio Layer-3)[10]形式にエンコードを行う。使用する音源はwavファイルのサイズ以外はすべて共通で、ビットレート1411kbps、オーディオサンプルレート44kHzの音声ファイルを使用する。表2に使用する音源を示す。なお音源は著作権フリーで提供されているwavファイル[19]を結合させて作成したオリジナル音源である。

表 2　使用するwavファイルのサイズ

使用するwavファイルのサイズ
１９５MB
３６６MB
７１１MB
１５００MB

ここでwavとは、Microsoftと IBM[14]により開発された音声データ記述のためのフォーマットである。通常は非圧縮でありデータ長が32bit符号なし整数型で記述されているため、4GBを超えるファイルは作成できない。今回使用するwavファイルは最大で約1GBのwavファイルである。※データ長を64bit符号なし整数型で記述するWave64というフォーマットも存在する。

また、mp3とはMPEG-1で利用されている音声圧縮方式である。

2.3 環境設定

2.3.1 MPICH2の導入

MPICH2[5]を使用するために、各計算機にWindows版の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のMPICH2の公式ページ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を行う。インストール手順を以下に示す。

①MPICH2公式ホームページよりWindows版MPIソフトmpich2-1.2.1-win-ia32.msiを”C:¥Program Files¥MPICH2”にインストールを行う。
②MPICH2のバイナリのある”C:¥Program Files¥MPICH2”¥bin”に対して環境変数PATHを指定する。
③ネットワークを通して共有できるフォルダを指定する。本研究では”C:¥mpi”を共有フォルダの設定を行った。

※この手順は並列計算をするすべてのPC同じ処理を行う。MPICH2を使用するに当たって並列計算に使用するPCは同一名の管理者権限のあるアカウント(英数字)および同一パスワードを使用する。

2.3.2 Microsoft Windows SDK の導入

Microsoft Windows SDK[11]とは、Windows で動作するアプリケーションを作成するために Microsoft が無料で公開しているソフトウェア開発キットであり、本研究では音声ファイルを処理するので Windows API を利用するため必要なヘッダファイル、ライブラリ、ツール、サンプルをインストールする。

インストール方法は Microsoft の公式ホームページから WebSetup 版もしくは ISO 版をダウンロードし C:¥Program Files¥Microsoft Platform SDK にインストールする。

2.3.3 Visual C++ 2005 Express Edition の導入

本研究では mp3 エンコーダを作成するにあたり C++[13]を使用した。Microsoft により Visual C++ Express Edition[12]は無料公開されている。インストール方法と環境設定の方法を以下に示す。

① Microsoft 公式ページより Visual C++ Express Edition をダウンロード、インストールを行う。
② "C:¥Program Files¥Microsoft Platform SDK¥Setup¥Register PSDK Directories with Visual Studio"を実行しパスを通す。
③ Microsoft Visual C++ 2005 Express Edition を起動し、[ツール]→[オプション] オプションダイアログを開く。ツリービューの[プロジェクトおよびソリューション]→[VC++ディレクトリ]を選択。 [ディレクトリを表示するプロジェクト]のコンボボックスで[インクルードファイル]を選択リストに以下を追加
   ✓ C:¥Program Files¥MPICH2¥include
④ 同様に[ライブラリファイル]に以下のパスを追加する。
   ✓ C:¥Program Files¥MPICH2¥lib
   ✓ C:¥Program Files¥Microsoft Platform SDK¥Lib

2.4 mp3 エンコーダ

本研究で使用するエンコーダはフリーで提供されている午後のこ～だ[15]で、lame と呼ばれるエンコーダを使っているダイナミックリングライブラリ gogo.dll 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。現在 gogo.dll は LPGL ライセンスによって配布されており、ソースコードのみの配布となっている。

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 をメモリへ読み込む。
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。

2.5　実行手順

MPICH2 では各プロセスに自動的にランクを割り当て、そこで各プロセスへの wav ファイルの振り分けはランクにより振り分けることができる。つまり、ランク n のプロセスに対しては”audio_n.wav”を振り分ければよいことになる。

本研究で作成した並列エンコーダの実行手順に以下に示す。また本研究の並列エンコードの実行概念図を図２に示す。

① 研究で作成した並列エンコーダの実行ファイル sotuken.exe を各計算機の”C:¥mpi”フォルダに置く。

② 入力となる wav ファイルを各計算機の”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。ランク n の計算機には、正常にエンコードすることができればフォルダに audio_n.mp3 と出力される。

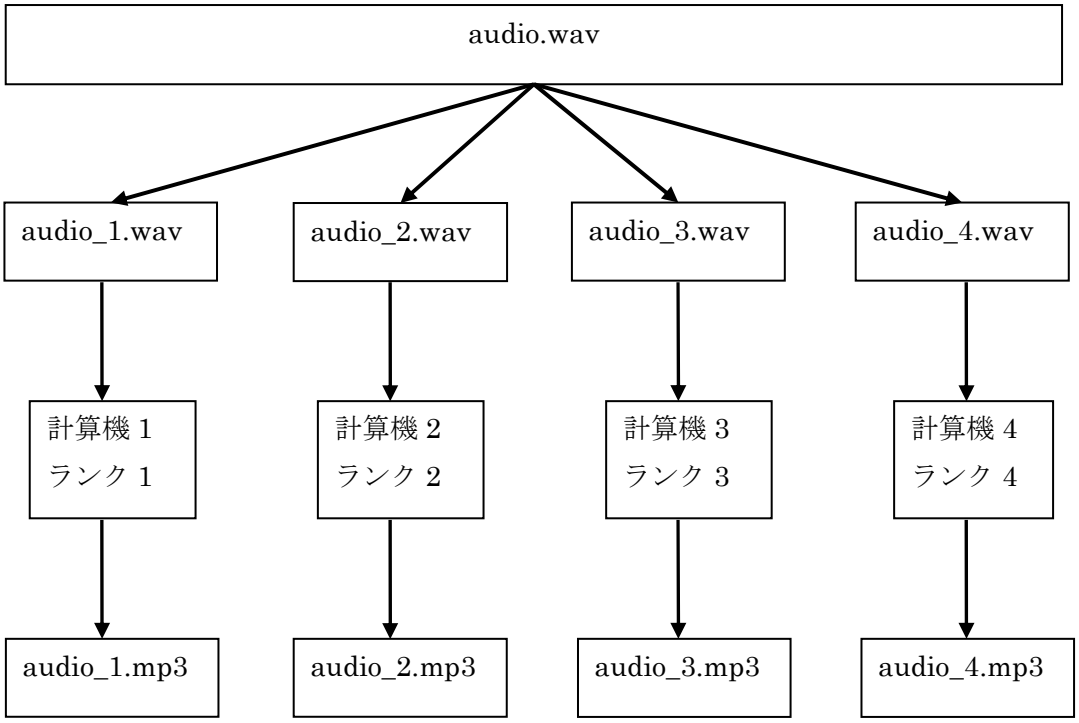

図 2　並列エンコードの実行概念図

3.　結果

本研究では、wav ファイルのサイズと並列計算機の台数を変えながら mp3 へエンコードに要した実行時間の測定を行った。誤差を考慮し、3 回計測を行い平均の値(少数第 3 位以下切り捨て)を用いる。本研究での測定結果を表 3 に示す。

表 3　実行結果の平均値(秒)

	1 プロセス	2 プロセス	4 プロセス
１９５MB	28.36	11.95	6.55
３６６MB	53.83	21.04	11.77
７１１MB	116.24	59.78	30.83
１５００MB	324.25	141.38	69.86

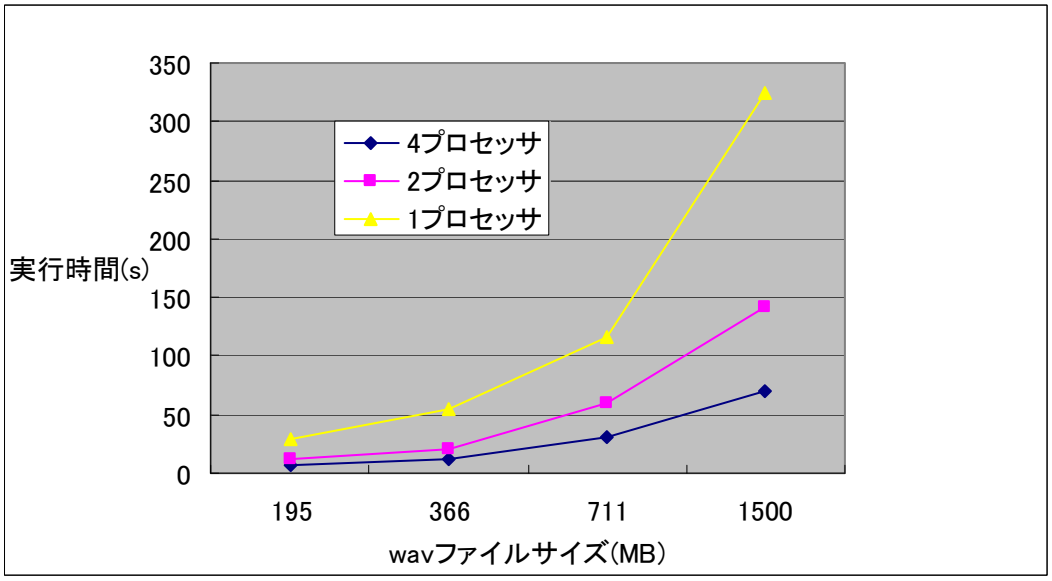

図 3　計算機の台数と実行時間

4.　結論・考察

表 3 よりわかるように、wav ファイルから mp3 ファイルへのエンコードでは並列計算をすることにより実行時間は短くなり、計算機の数が増えれば増えるほど実行時間が短縮されていることが示される。従って、MPI による仮想並列計算環境の構築は非常に有用であると考えられる。

しかし、本研究ではファイルは予め並列計算するファイルを分割しているため、分割にかかる時間を考慮していないということが挙げられる。したがって、ファイル分割にかかる時間も含めて並列処理の有用性を検証することが今後の課題である。また、本研究ではCPUの性能差を考慮せず、データを均等分別しており、考慮して負荷分散をすることが今後の課題である。

謝辞

本研究を行うにあたり協力者のみなさん、様々な助言を頂いた石水隆先生に感謝の意を表します。ご迷惑も沢山お掛けしましたが一年間本当にありがとうございました。

参考文献

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

[2] 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　著：W.グロップ,E.ラスク,T.タークル　訳：畑崎隆雄 ：実践ピアソン・エデュケーション(2002)

[3] MPI による並列プログラミングの基礎　著：渡邊真也
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] OAK RIDGE National Laboratory :http://ww.ornl.gov/

[9] ディジタル・サウンド処理入門　著：青木直史(2006)

[10] たちまちわかる MP3　著：大沢文考：工学社(1999)

[11] Microsoft Windows SDK http://msdn.microsoft.com/ja-jp/windows/bb980924.aspx

[12] VisualStudio2005ExpressEditions :http://www.microsoft.com/japan/msdn/vstudio/express/default.aspx

[13] Visual C++ 2005 ビギナー編 著：林 晴比古

[14] IBM：http://www.ibm.com/jp/

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

[16] Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing：http://www.netlib.org/pvm3/book/pvm-book.html

[17] 並列計算機アーキテクチャ　著：奥川峻史(1991)

[18] J.JaJa：An Introduction to Parallel Algorithms ,Addison Wesley(1992)

[19] フリー音源-Sound ever-：http://www.yamasen-e.net/freemusic/index.html

付録　並列エンコーダ実行ファイル作成プログラムのソースコード

①　encoder.cpp

```
#define MPICH_SKIP_MPICXX
#include "mpi.h"
#include <stdio.h>
#include <windows.h>
#include "musenc.h"
#include <time.h>
#include <stdlib.h>

#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;
```

```
}
```

① stab.ccp

```
#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 );
}
```

③ musenc.h

```
/* -*- TABSIZE = 4 -*- */
```

```
/*
 *      for new GOGO-no-coda ( 2000/1/15 )
 *      Copyright (C)1999,2000 PEN@MarineCat
 */
#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 unknow
                // 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__ */
```