卒業研究報告書

題目

# ＭＰＩを用いた最適な分散処理

指導教員

石水 隆 助教

報告者

05-1-37-0136

角仁志

近畿大学理工学部情報学科

平成 21 年 1 月 31 日提出

# 概要

計算機は常に高い処理性能が求められている。処理性能の向上の手段として、限界まで１台の計算機の処理性能を向上させる方法と、複数台の計算機を使用しての並列計算がある。しかし、１台の処理性能の向上には理論上の限界があり、また、性能の向上には時間と資金がかかってしまう。そこで、もう１つの手段である並列計算が重視されている。だが、高い処理性能を持つ並列計算機はとても高価なものであり、メリットが少ない。そのため、ネットワークを利用することで複数の安価な計算機を並列計算機として利用できるソフトウェアが注目されている。

本研究では、無料で提供されている並列計算が可能となるソフトウェアの１つである MPI(Message Passing Interface)を用い、通常ならば均等に分割されるデータの割合を、計算機の性能により変動させることで、均等に分割された場合との処理時間を比較し、最適な分割方法を実験的に評価する。評価方法として、視覚的に分割割合の変動を捕らえるのが可能なことから、無圧縮 BMP(Bit Map)画像から JPEG(Joint Photographic Experts)画像に変換処理を行い、どれだけの処理時間の短縮が出来るのか検証を行なう。

# 目次


1 序論 ..... 3

1.1 本研究の背景 ..... 3
1.1.1 並列処理 ..... 3
1.1.2 並列計算機 ..... 2
1.1.3 PVM ..... 2
1.1.4 MPI ..... 3
1.1.5 PVM と MPI の比較 ..... 3
1.2 本研究の目的 ..... 3
1.3 本報告書の構成 ..... 4

2 準備 ..... 5

2.1 使用機器 ..... 5
2.2 MPICH2 の導入 ..... 6
2.3 画像ファイル ..... 6
2.3.1 BMP ..... 6
2.3.2 JPEG ..... 7
2.3.3 JPEG エンコード ..... 7
2.3.4 本研究で使用する画像 ..... 7
2.4 JPEG 並列エンコーダ ..... 7

3 計測結果 ..... 8

3.1 全体の処理時間 ..... 9

4 考察 ..... 11

5 結論・今後の課題 ..... 12

謝辞 ..... 13

参考文献 ..... 14

付録 ..... 15


# 1 序論

## 1.1 本研究の背景

### 1.1.1 並列処理

一つのタスクにおいて複数のプロセッサを用い、分割して処理を行なう事により、単一のプロセッサでの処理よりも高速に計算処理を行なうことを並列処理(Parallel Processing)という。

近年、計算機の性能の向上は著しいが、いずれ限界が来ることはすでに指摘されており、単一のプロセッサの性能に拠ることなく、処理速度を向上させる手段として並列処理が挙げられる。すでに並列処理は天体の軌道観測や地球の気象シミュレーションなどの逐次処理では膨大な時間がかかる問題におい

て、逐次計算機よりも短時間で解けることから利用されている。

### 1.1.2 並列計算機

一つのタスクにおいて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。

並列計算機には､共有メモリ型並列処理と分散メモリ型並列処理の 2 つに大きく分けることが出来る。

共有メモリ型並列処理とは、全てのプロセッサが一つのメモリを共有して使用する。この方法では同期の問題やデータの送受信などのオーバヘッドにはメモリを共有しているため容易に対処できる。しかし、プロセッサ数の増減などの変化に対して、全てのプロセッサをメモリに接続させることが困難である｡共有メモリ型並列処理モデルとして PRAM(Parallel Random Access Machine)[4]などが挙げられる。

分散メモリ型並列処理とは、それぞれのプロセッサがここに局所メモリを持ち、プロセッサ間の通信にはネットワークを使用する。この方法で各プロセッサの同期の問題やデータの送受信時間がそのままオーバヘッドとして処理時間に影響してしまう。しかし、現在は共有メモリ型並列処理の効率的な実装が困難なことから、実装が容易な分散メモリ型並列処理が主流となっている。また、分散メモリ並列処理モデルとして BSP(Bilk-Synchronous Parallel)[14]などが挙げられる。

共有型並列計算機,分散メモリ型並列計算機共に現存する様々な並列計算機が存在する。しかし、一般的に並列計算機は非常に高価であり、並列計算機を持つことができるのは一部の大学、研究機関、そして企業など、資金力を持つ組織に限られる。このため、多くのユーザは手軽に並列計算機を使うことができず、並列計算機は注目されるものの広く普及されることはなかった。だが今日、複数の計算機をネットワーク接続し、計算機群全体を 1 台の仮想並列計算機とするクラスタ(Cluster)環境が、注目されておりクラスタ環境を構築するソフトウェアも開発されている。

クラスタ環境を構築するソフトウェアの中には、無料で提供されているものもあるため、現在では多くのユーザが並列計算機を身近に利用できるようになった。そのようなソフトとしては共有メモリ型並列処理用として OpenMP[6],分散メモリ型並列処理用として MPI[2][3]や PVM[5]といったものがある。

以下に分散メモリ型並列処理用である MPI と PVM の特徴について述べる

### 1.1.3 PVM

PVM(Parallel Virtual Machine)[5]は、1991 年に米国のオークリッジ国立研究所(Oak Ride National Laboratory)[13]を中心に異機種間の分散処理が目的に開発された、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 TCP/IP の通信ライブラリであり、一般的に使用されている LAN 環境があれば並列処理を利用することが可能であり、多くのユーザに普及している。また、異機種間の通信、一般に普及しているパーソナルコンピュータから一部の大学や研究機関などしか持っていないスーパーコンピュータなどにも対応しており、機種に関係なく PVM を利用することで並列処理を可能としている。

PVM とは、計算機上にデーモンと呼ばれるものを存在させ、このデーモンを用いて各計算機間の通信を行うことによりデーモンで接続された複数の計算機を一つの仮想並列計算機として構成させる。そしてユーザは PVM アプリケーションを一人で複数実行することも可能である。そして、PVM インターフェイスルーチンライブラリでメッセージパッシング、プロセスの生成、タスクの協調、仮想計算機の構成ルーチンを提供している。

PVM の特徴は耐故障性の高いことにある。通常なら仮想並列計算機の演算中にどれか一台でも接続されている計算機が停止、もしくは接続が途切れてしまうと、計算処理にエラーが出てしまうが、PVM は任意で計算機の追加や削除が行なえ、故障した計算機を仮想並列計算機内から迅速に削除し、計算処理を停止することなく続けることができる。

PVM の問題点は移植性に乏しいことにある。PVM は各並列計算機ベンダが独自にチューニングを行っており、それぞれが独自の PVM を開発してしまった。そのため PVM で作成をしたプログラムを他の仮想並列計算機に移植することが難しい。これは、第三者機関によるしっかりとした規約を設けなかったことに起因している。

### 1.1.4 MPI

MPI(Massage Passing Interface)[2][3]は並列計算におけるメッセージ（データ）通信のプログラムを記述するための規約を設けるために開発された。名前の通り PVM のようにソフトウェアがあるというわけではない。高速な伝送と PVM 同等の利便性を有するインタフェースである。

MPI は 1992 年に結成された MPI フォーラムにより、並列処理に関する標準的使用の定義、および検討をされることで具体化された。MPI は MPI 標準と呼ばれるインタフェース規格であり、この MPI 標準に準じて実装されたライブラリを MPI ライブラリと呼ぶ。MPI の開発には欧米の 40 近い組織から 60 人ほど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。MPI は PVM よりも後に開発され、PVM と同等の機能を持ち、問題点の改善も含まれている。

本来、計算機間で通信を行う場合、計算機のアーキテクチャによりプロトコルの違いが生じ、計算機ごとにプログラムを書き分ける必要があった。しかし、MPI ライブラリには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 MPI によりプロトコルの障害を考慮することなく、ユーザは並列処理アルゴリズムに集中してプログラム作成することが出来る。

PVM と違う点は、異機種間の通信が考慮されていないことにある。MPI を用いて仮想並列計算機を構築するには、使用する計算機のオペレーティングシステムを統一なければならないという制約が存在し、実行時の計算機の台数も固定化されている。そのために PVM よりも耐故障性は低い。

MPI は専用の並列計算機からワークステーションやパーソナルコンピュータなど多用な機種をサポートしており、無料で提供されているものも多く、主な実装に MPICH[8][9]や LAM[11][12]がある。また、MPI を使用できるプログラミング言語も多く、C 言語や Java など多様に対応している。

### 1.1.5 PVM と MPI の比較

PVM と MPI の大きな違いとして移植性にある。

MPI は PVM と違い世界的な標準が目的に開発されたものであり､その移植性は高く評価されている。MPI 規格を用いて開発されたプログラムは通信プロトコルに拠らずに並列処理アルゴリズムが設計できる。PVM は各並列計算機ベンダが独自にチューニングを行っており、作成をしたプログラムを他の仮想並列計算機に移植することが難しい設計となっている。

MPI の欠点は､仮想並列計算機を構成する計算機のオペレーションシステムを統一しなければ動作がしないため、異機種間で並列処理ができない。PVM は MPI と違い異機種間における処理にも対応して作成されているので、異機種の計算機による並列処理にも用いることができる。

以前は PVM と MPI の比較として動的なプロセス管理があったが、MPI の新規格である MPI-2 の仕様において動的なプロセス管理の機能が取り入れられた｡これにより MPI でも動的にアプリケーションの中からプロセスを生成、停止などできるようになった。これにより PVM の優位性の 1 つが失われたことになる。

現在の両方の開発状況は、調べた中で PVM の開発は昔ほどに進んでいないように思われる。逆に、MPI は現在でも研究開発に行なわれており、MPI-1.3 や MPI-2.2、MPI-3.0 などが開発中である。

そこで本研究では、MPI を使用する。

## 1.2 本研究の目的

一般に MPI を構成する計算機は、その性能が均一ではなく様々な CPU 速度やメモリ容量を持っている。このため、各計算機に処理をさせる場合、計算機ごとに処理速度が異なる。そのため、一部の計算機が極端に遅い場合、その計算機の処理の終了まで他の計算機が処理を待つ必要があるので、全体の処理速度が低下する。よって、処理性能の異なる計算機群を用いて、効率よく並列処理を行なうには、計算機の性能に応じて、処理を割り当てなければならない

そこで本研究では、MPI による仮想並列計算機において、計算機の性能によりデータ分割割合を変動させた時の処理速度がどのように変化するかを測定し、最適なデータ分割の方法を検証する。評価方法

として視覚的に分割割合を捉えることができるため画像データを使用し、BMP 画像から JPEG 画像にエンコードさせる並列エンコーダを使用し、過程で均等に分割される画像の割合を変動させ、4 台の計算機で均等分割した場合の処理時間と、性能により分割の割合を変動させた場合の処理時間を比較し、並列化により処理速度の向上が得られるか検証を行なう。

## 1.3 本報告書の構成

本報告書の構成を以下に述べる。2 節に本研究での環境と、使用する JPEG エンコーダについて述べる。3 節に計測結果。4 節では考察。5 節では本研究の結論と今後の課題を述べる。

# 2 準備

## 2.1 使用機器

本研究では、仮想並列計算機を構築するため、MPICH2[8][9]というフリーソフトウェアをダウンロードし、インストールと環境設定を行う必要がある。MPICH2 を使用するのは、無料であり広く普及しているソフトウェアであること、MPI の特徴でもある移植性の高さをそのまま利用できる、MPI-2 の機能に対するサポートが充実している、Windows のオペレーションシステムシリーズでも利用することができる、などの理由からである。

本研究で使用する計算機では、マイクロソフト社の提供販売しているオペレーションシステムである Windows XP を使用する。Windows XP を使用する理由は、一般家庭や公共機関、企業などの場所で他のオペレーションシステムより広く普及しており、マイクロソフト社が新しく提供するオペレーション Windows Vista よりも対応するソフトウェアが多いからである。また、MPI を使用して仮想並列計算機を構築する場合に、全ての計算機が同じオペレーションシステムを実装しておかなければ実行できないこともあり、オペレーションシステムの統一が容易であるという理由もある。

本研究において、使用する計算機の名前、性能などを表 1 に記す。

表 1：使用する計算機の構成

| | OS | CPU | メインメモリ |
|---|---|---|---|
| 計算機 F8 | Microsoft Windows XP | Intel Pentium(R) 1.5GHz | 504MB |
| 計算機 FM | Microsoft Windows XP | Intel Pentium(R) 3.2GHz | 2000MB |
| 計算機 M2 | Microsoft Windows XP | Intel Celerpn(R) 2.5GHz | 1000MB |
| 計算機 I5 | Microsoft Windows XP | Intel Pentium(R) 1.9GHz | 640MB |

また、本研究にはこれら計 4 台の計算機での実験を行なう。仮想並列計算機を構築する際、LAN やルータ、ハブを使用してネットワークを構成している。本研究で使用した仮想並列計算機の構成図を図 1 に示す。

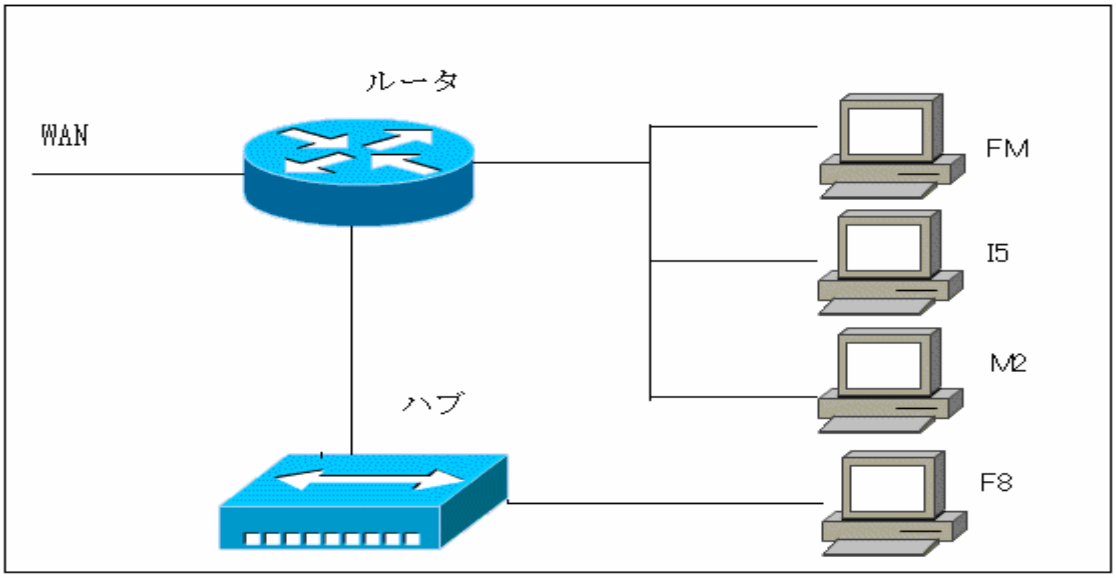


図 1：仮想並列計算機の構成図

## 2.2 MPICH2 の導入

以下に MPICH2 のインストール手順を述べる。

まず MPICH2 のページから使用する OS に合うソフトウェアをダウンロードする。本研究では WindowsOS を使用したので、Windows 用の最新バージョンを用いた。

ダウンロード後、exe ファイルを実行することにより、インストーラが起動する。

インストール後は環境設定を行なわなければならない。この設定は環境変数 PATH の指定を、MPICH2 のインストールしたフォルダの下にある bin フォルダに対して設定すればよい。この設定の確認は PATH 指定を行なった後、コマンドプロンプトで mpiexec コマンドを実行したときに”Usage”と表示されれば、PATH 指定が成功したことがわかる。

また、使用する計算機に共通のアカウント名とパスワードを持つユーザを用意する必要がある。加えてプログラムの実行には、全ての計算機に共通したファイル構成で、実行ファイルを置く必要がある。そのため、実行ファイルの受け渡しなどをネットワーク上で行えるようにするため、共有フォルダを用意することが望ましい。

今回使用した並列 JPEG エンコーダは C 言語で作成されており、コンパイルツールにはマイクロソフト社の Visual C++2005[7]を使用した。MPICH2 を使用できるように設定するには、ツールオプションからインクルードファイルとライブラリファイルに MPICH2 フォルダの lib と include フォルダを追加する。次に実行するプロジェクトのプロパティ設定で、リンカ、入力の依存ファイルに mpi.lib と cxxd.lib を追加する必要がある。

## 2.3 画像ファイル

### 2.3.1 BMP

BMP(BitMap)[10]とは、Windows で最も標準的な画像フォーマットである。BMP ファイルは、ヘッダ部とデータ部にデータが分かれており、ヘッダ部に画像イメージの色数や、大きさ、データ部に画像イメージが保存されている。Windows の標準ファイル・フォーマット形式なので Windows 環境であるなら、簡単な画像データのやり取りに最も適している。しかし、データをほとんど圧縮させずに保存するので、ファイルサイズが大きくなりやすい。

### 2.3.2 JPEG

JPEG(Joint Photographic Experts Group)[10]とは静止画像データの圧縮方式の一つであり、ISO により設置された専門家組織である　”Joint Photographic Experts Group”の頭文字を繋げたものである。圧縮の際に画質を劣化させてデータの容量を削減する方式と、劣化させない方式が選べ、画質を劣化させる場合はどの程度劣化させるか指定留守ことができる。256 色の GIF(Graphics Interchange Format )[10]とは違い、フルカラーにも対応しており写真などの圧縮に適している。現在、GIF と並びホームページ上で使用される画像データ形式の標準として使われている。

### 2.3.3 JPEG エンコード

JPEG の符号化処理の特徴は、ブロック分割、DCT 変換、量子化である。

ブロック分割とは、入力された画像データを左上端から右端までサンプリングし、それが終わると下段に行き、同じように左端から右端へと進んでいく、その過程で 8×8 画素で構成されるブロックの集合の MCU と呼ばれる単位まで分割する。JPEG の符号化では MCU の単位で処理が行われるため、本研究では、それぞれサイズ 8×8 の定数倍である部分画像を一つの入力画像とし、各計算機の性能に比例する個数の基本画像から成る部分画像に分割し、各部分画像を各計算機に割り当てる。

DCT 変換(離散コサイン変換)では、複雑な情報で高背されている画像データを、単純な空間周波数成分に分解して変換を行なう。この作業により画像データの容量を削減することが可能となる。しかし空間周波数成分の係数を実数で処理しようとすると、小数点以下のデータが残り、処理効率が悪くなるため、ハフマン符号化を行なっている。

量子化とはデータ量の係数を整数に変換する処理であり、量子化処理は量子化マトリクスに基づいて輝度と色素の 1 ブロック単位で行なわれる。この作業により共通するデータを増やし、画像データ容量の削減をする。

最後に処理効率を向上させるためにハフマン符号処理を行う。ハフマン符号処理は長いビット列と短いビット列に割り当てをして、圧縮していく。

### 2.3.4 本研究で使用する画像

本研究では分割の割合の変動を視覚的に捕らえられるため画像データを使用している。使用した画像は無圧縮の BMP 画像である。BMP 画像は 24bit のフルカラーでサイズが 2560×1920、データの容量が約 14MB で統一している。

使用する画像の枚数は 50 枚。実行の際には、1･5･10･25･50 と順に計測を行い、各枚数と割合別の処理時間を計測する。

## 2.4 JPEG 並列エンコーダ

本研究で使用する JPEG 並列エンコーダは､MPI を用いて仮想並列計算機を構成している全ての計算機に指示を行い、BMP 画像を JPEG 画像にエンコードを行う。また指示を送る計算機をメイン計算機としている。

メイン計算機は、BMP 画像を読み込み画像に分割する。BMP 画像を分割する際にメイン計算機は、仮想並列計算機を構成しているサブ計算機から、計算処理を行なう際に使用する計算機の台数を取得することで BMP 画像の分割を行なう。本研究ではこの時に各計算機の性能を取得し、その性能に従い分割の割合を求めて分割することを目標としていたが、性能の取得と性能により動的に分割するプログラムが非常に困難だったため､BMP 画像の分散はプログラムの実行前に行ない BMP 画像の参照パスを変更することで、目的だった画像データの分割の割合を変更した場合の処理時間を出している。

## 3　計測結果

本研究での比較・検証では、BMP 画像の枚数を 1･5･10･25･50 の順にエンコード処理を行なう。その際に分割するデータの割合を計算機の CPU のクロック数の比、メインメモリのバイト数の比、CPU のクロック数×メインメモリのバイト数の比の四つの割合で分割し、計 4 回の計測を行い、それぞれの処理時間を比較した。

図 2.1、図 2.2、図 2.3 にデータを計算機の性能別に分割した割合を示す。ただし均等に分割した場合の図は省略する。

また、JPEG 画像は縦横のサイズが 8×8 の画像を基本画素とするため、画像サイズが 8×8 の倍数ではない場合、処理時間が増加する可能性がある。そこで本研究では、入力画像を 8×8 の倍数サイズを持つ 64 の基本画像に分割し、その基本画像で構成される部分画像を各計算機に計算機の性能に比例するように割り当てた。

表 2 に各計算機に割り当てた基本画像数を示す。

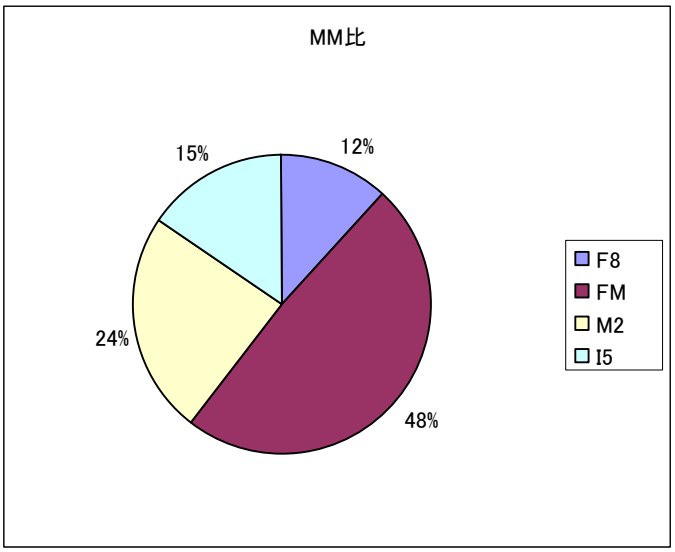


図 2.1：各計算機のメインメモリのバイト数の比

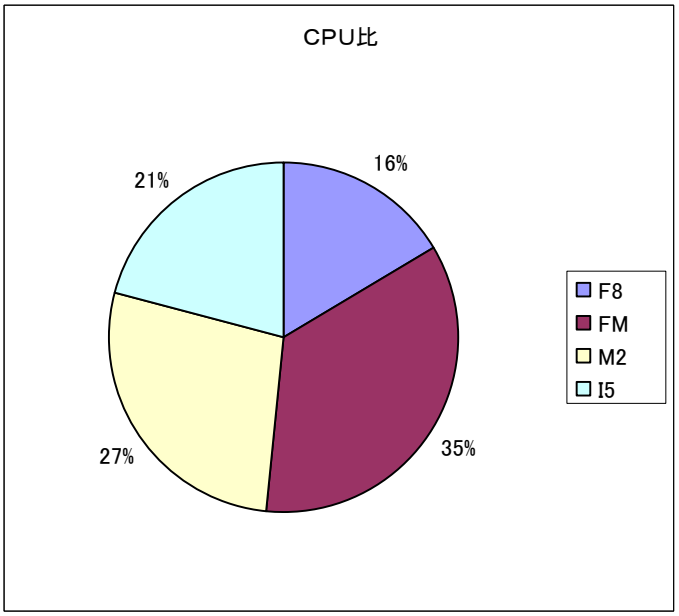

図 2.2：各計算機の CPU のクロック数の比

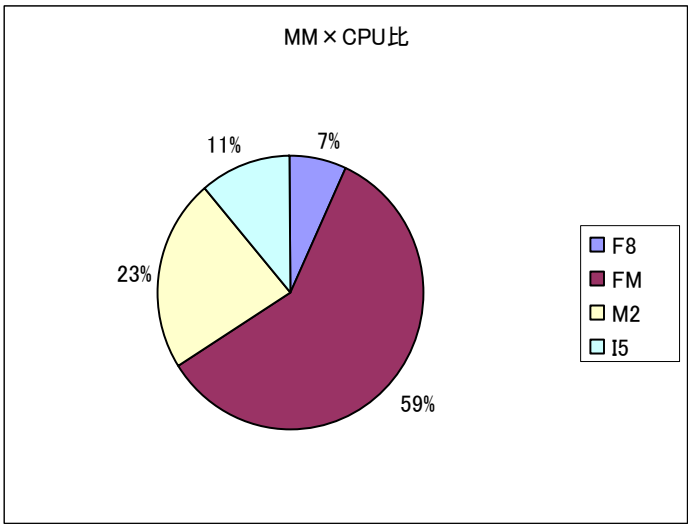


図 2.3：各計算機のメインメモリのバイト数×CPU のクロック数の比

表 2 計算機への基本画像の割り当て数

| | MM 比 | CPU比 | MM×CPU 比 |
|---|---|---|---|
| F8 | 8 | 10 | 4 |
| FM | 31 | 22 | 38 |
| M2 | 15 | 17 | 15 |
| I5 | 10 | 13 | 7 |
| 合計 | 64 | 64 | 64 |

## 3.1 全体の処理時間

ここでは、BMP 画像データのエンコード処理においての演算時間だけを示す。

表 3 および図 3 に BMP 画像データのエンコード処理の実行に要した演算時間を示す。

この計測結果を見ても分かるように、処理における演算時間自体は、ほぼ使用している CPU の数だけ反比例して減っている結果が得られた。

表 3 各割り当てのエンコード処理時間(秒)

| | 1 | 5 | 10 | 25 | 50 |
|---|---|---|---|---|---|
| 均等 | 2.511952 | 14.21708 | 26.82738 | 64.36863 | 131.3696 |
| CPU | 2.261913 | 11.55877 | 23.11291 | 57.78268 | 116.0681 |
| MM | 3.095983 | 15.95123 | 32.1589 | 81.00014 | 162.9415 |
| CPU×MM | 3.655275 | 18.51377 | 36.88822 | 92.58623 | 185.2456 |

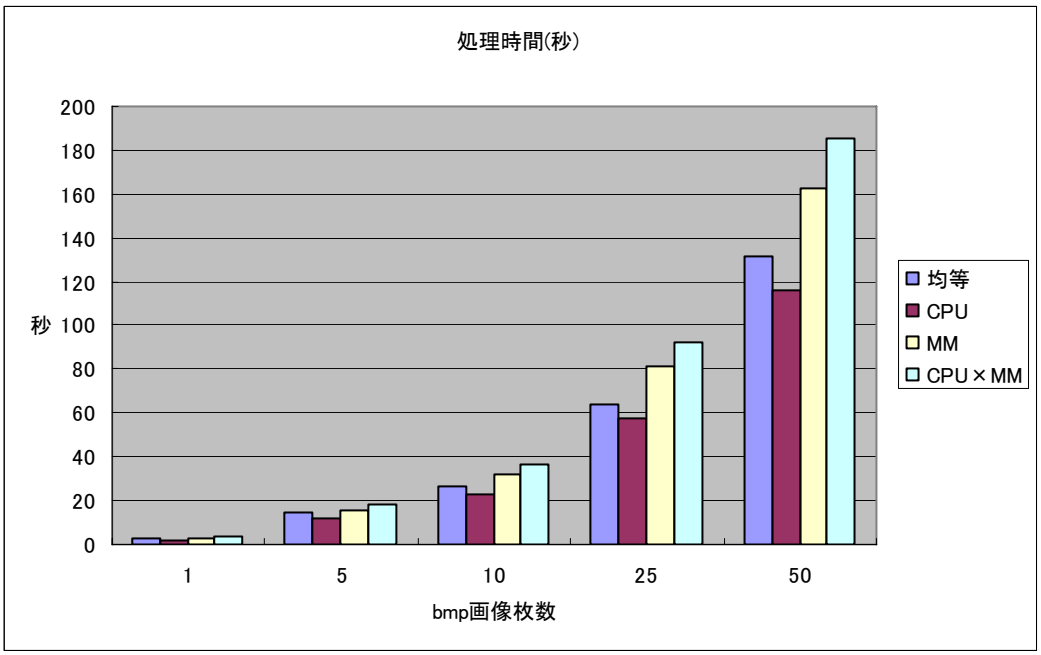


図 3：各割り当てのエンコード処理時間(秒)

# 4　考察

本研究で得られた計測結果では、CPU のクロック数の比で分割した場合に処理速度の向上が確認できた。しかし、メインメモリのバイト数の比およびメインメモリのバイト数×CPU のクロック数の比では、データを均等に分割した場合に比べて著しく悪化している。これは今回の処理に関してメインメモリを使用がそれほど大きく関係していなかったため、CPU のクロック数の比較が適した分割方法だったと考えられる。

また、個々の計算機の演算時間を比べると最適にデータの割合を分割したにも関わらず、演算時間に差が見られた。もしも理想的なデータの分割が為されていたならば、全ての演算時間が同じ、または無視してもかまわない程度の差しか出ないはずである。そのため、CPU のクロック数に依存する方法も最適とは言えない。

# 5　結論・今後の課題

本研究では並列処理において均等に分割されるデータの割合を変動させることで、処理速度を向上させることが示した。しかし、不適切な割合を分割すれば、均等に分割した場合よりも処理速度が低下することも検証結果として確認できる。

本研究ではデータを分割するのに最適な割合を求めたとは言えない。これは求められる処理や接続されるネットワークの速度により､使用されるメインメモリのバイト数も変化することが考えられるので、様々な処理においての検証が必要であり、今回の検証結果だけでは不十分である。また、本研究では使用したプログラムでは接続される計算機の性能を自動的に出力し、割り出した性能によりデータ分割割合を変動させることができていない。本来なら不特定多数の計算機と接続して仮想並列計算機とすることもあるので、全てをプログラムにより処理できなければ実用性に欠ける。そして、プログラムの問題が解決したとしても、根本的な部分で重大な問題が残る。それは処理工程において、一度接続される全ての計算機と同期し、個々の計算機の性能を出力すること工程と、その出力結果に依存してデータを分割する工程が、通常の方法よりも余分にオーバヘッドとして存在することである。これにより、この方法は膨大な処理かつ計算機の性能差が大きい場合は有効だと考えられるが、それ以外の場合ではたとえ最適な分割割合が求められても結果として処理時間が増加してしまうことが考えられる。

したがって、今後の課題としてはオーバヘッドをなくすため、仮想並列計算機に使用される計算機の性能を処理実行時に求めるのではなく、接続した段階で計算機の性能を把握できるシステムを構築することが考えられる。

## 謝辞

本研究を完成するにあたって、夜遅くまでご指導していただいた石水助教、また同研究室の皆さんには様々な助言をしていただき感謝の言葉を述べたいと思います。

# 参考文献


[1] 横瀬拓也：ＭＰＩによる JPEG 圧縮の検証, 卒業研究報告書(2006).
[2] 秋葉博(訳),P・パチェコ(著)：MPI 並列プログラミング、倍風館(2001)
[3] 畑崎隆雄(訳), ラジーブ・タークル,ユーイング・ラスク,ウイリアム･グロップ(著)： 実践 MPI-2. ピアソン・エデュケーソン(2002)
[4] Josepb JaJa(著)：AnIntroduction toParallelAlgorithms,Addison Wesley Publishing Company
[5] Al Geist et al.,"PVM:Parallel Virtual Machine,"MIT Press1994
[6] 牛島省(著)：OpenMP による並列プログラミングと数値計算法、丸善株式会社(2006)
[7] Visual Studio 2005.(URL)： http://www.microsoft.com/japan/msdn/vstudio/
[8] MPICH2 (URL) : http://www.mcs.anl.gov/research/projects/mpich2/downloads/index.php?s=downloads
[9] MPICHonWindowsLocal(URL):
http://ums.futene.net/wiki/Paralell/4D5049434832206F6E2057696E646F7773204C6F63616C.html
[10] 音声・動画・文書ファイル形式の達人になる本、工学社(2001)
[11] MPI ライブラリの調査と性能の計測(URL)：
http://mikilab.doshisha.ac.jp/dia/research/report/2004/0613/001/report20040613001.html
[12] MPI 覚え書き[YASUAKI ITO's HomePage](URL)：
http://www.cs.hiroshima-u.ac.jp/~yasuaki/dokuwiki/doku.php?id=mpi:mpi
[13] URL：http://www.ornl.gov/
[14] A.Baumker,W.Dittrich,F.M.Heide and I.Rieping,"Realistic Parallel Algorithms:Priorit Queue Operations and selection for BSP Model," Proc.EuroPar'96,VOl.II.pp.369-376,1996.

# 付録

以下に本研究で使用した並列 JPEG エンコーダプログラムおよび BMP 画像入出力プログラムを示す.

1. jpgcs.c
2. bmp.h
3. bmp.c

jpgcs.c

```
/*
 * JPEG encoding engine for DCT-baseline(Not JFIF)
 *
 *      copyrights 2003 by nikq | nikq::club.
 */



#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"

//DCT::Chen DCT に依存

#include "chendct.c"
int K=0;
int myid;
unsigned char *local_b;
#define qua 100

#define N 5

//ISO/IEC 10918 ITU-T 勧告 T.81 付属書 K
//輝度用量子化表(ジグザグシーケンス順)
unsigned char jpeg_qt[] = {
    0x10,0x0B,0x0C,0x0E,0x0C,0x0A,0x10,0x0E,
    0x0D,0x0E,0x12,0x11,0x10,0x13,0x18,0x28,
    0x1A,0x18,0x16,0x16,0x18,0x31,0x23,0x25,
    0x1D,0x28,0x3A,0x33,0x3D,0x3C,0x39,0x33,
    0x38,0x37,0x40,0x48,0x5C,0x4E,0x40,0x44,
    0x57,0x45,0x37,0x38,0x50,0x6D,0x51,0x57,
    0x5F,0x62,0x67,0x68,0x67,0x3E,0x4D,0x71,
    0x79,0x70,0x64,0x78,0x5C,0x65,0x67,0x63,

    0x11,0x12,0x12,0x18,0x15,0x18,0x2F,0x1A,
    0x1A,0x2F,0x63,0x42,0x38,0x42,0x63,0x63,
    0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x63,
    0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x63,
    0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x63,
    0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x63,
    0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x63,
```

```
    0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x63
    };


// ハフマンテーブル
// http://www.geocities.co.jp/SiliconValley-SanJose/8609/labo/jpegcoder.html
//  jpegcoder.java より、引用

// 輝度 DC
unsigned char ht0[] = {
    0x00,0x01,0x05,0x01,0x01,0x01,0x01,0x01,
    0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,
    0x08,0x09,0x0A,0x0B
    };
unsigned char  hsizeT0[] = {
    2,3,3,3,3,3,4,5,6,7,8,9
    };
int hcodeT0[] = {
    0x0000,0x0002,0x0003,0x0004,0x0005,0x0006,0x000e,0x001e,
    0x003e,0x007e,0x00fe,0x01fe
    };

//色差 DC
unsigned char  ht2[] = {
    0x00,0x03,0x01,0x01,0x01,0x01,0x01,0x01,
    0x01,0x01,0x01,0x00,0x00,0x00,0x00,0x00,
    0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,
    0x08,0x09,0x0A,0x0B
    };
unsigned char  hsizeT2[] = {
    2,2,2,3,4,5,6,7,8,9,10,11
    };
int hcodeT2[] = {
    0x0000,0x0001,0x0002,0x0006,0x000e,0x001e,0x003e,0x007e,
    0x00fe,0x01fe,0x03fe,0x07fe
    };

//輝度 AC
unsigned char  ht1[] = {
    0x00,0x02,0x01,0x03,0x03,0x02,0x04,0x03,
    0x05,0x05,0x04,0x04,0x00,0x00,0x01,0x7D,
    0x01,0x02,0x03,0x00,0x04,0x11,0x05,0x12,
    0x21,0x31,0x41,0x06,0x13,0x51,0x61,0x07,
    0x22,0x71,0x14,0x32,0x81,0x91,0xA1,0x08,
    0x23,0x42,0xB1,0xC1,0x15,0x52,0xD1,0xF0,
    0x24,0x33,0x62,0x72,0x82,0x09,0x0A,0x16,
    0x17,0x18,0x19,0x1A,0x25,0x26,0x27,0x28,
```

```
    0x29, 0x2A, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39,
    0x3A, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49,
    0x4A, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59,
    0x5A, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69,
    0x6A, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79,
    0x7A, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89,
    0x8A, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98,
    0x99, 0x9A, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7,
    0xA8, 0xA9, 0xAA, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6,
    0xB7, 0xB8, 0xB9, 0xBA, 0xC2, 0xC3, 0xC4, 0xC5,
    0xC6, 0xC7, 0xC8, 0xC9, 0xCA, 0xD2, 0xD3, 0xD4,
    0xD5, 0xD6, 0xD7, 0xD8, 0xD9, 0xDA, 0xE1, 0xE2,
    0xE3, 0xE4, 0xE5, 0xE6, 0xE7, 0xE8, 0xE9, 0xEA,
    0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7, 0xF8,
    0xF9, 0xFA
    };
unsigned char  hsizeT1[] = {
     4,  2,  2,  3,  4,  5,  7,  8, 10, 16, 16,  4,  5,  7,  9, 11,
    16, 16, 16, 16, 16,  5,  8, 10, 12, 16, 16, 16, 16, 16, 16,  6,
     9, 12, 16, 16, 16, 16, 16, 16, 16,  6, 10, 16, 16, 16, 16, 16,
    16, 16, 16,  7, 11, 16, 16, 16, 16, 16, 16, 16, 16,  7, 12, 16,
    16, 16, 16, 16, 16, 16, 16,  8, 12, 16, 16, 16, 16, 16, 16, 16,
    16,  9, 15, 16, 16, 16, 16, 16, 16, 16, 16,  9, 16, 16, 16, 16,
    16, 16, 16, 16, 16,  9, 16, 16, 16, 16, 16, 16, 16, 16, 16, 10,
    16, 16, 16, 16, 16, 16, 16, 16, 16, 10, 16, 16, 16, 16, 16, 16,
    16, 16, 16, 11, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
    16, 16, 16, 16, 16, 16, 16, 11, 16, 16, 16, 16, 16, 16, 16, 16,
    16, 16
    };
int hcodeT1[] = {
    0x000a, 0x0000, 0x0001, 0x0004, 0x000b, 0x001a, 0x0078, 0x00f8,
    0x03f6, 0xff82, 0xff83, 0x000c, 0x001b, 0x0079, 0x01f6, 0x07f6,
    0xff84, 0xff85, 0xff86, 0xff87, 0xff88, 0x001c, 0x00f9, 0x03f7,
    0x0ff4, 0xff89, 0xff8a, 0xff8b, 0xff8c, 0xff8d, 0xff8e, 0x003a,
    0x01f7, 0x0ff5, 0xff8f, 0xff90, 0xff91, 0xff92, 0xff93, 0xff94,
    0xff95, 0x003b, 0x03f8, 0xff96, 0xff97, 0xff98, 0xff99, 0xff9a,
    0xff9b, 0xff9c, 0xff9d, 0x007a, 0x07f7, 0xff9e, 0xff9f, 0xffa0,
    0xffa1, 0xffa2, 0xffa3, 0xffa4, 0xffa5, 0x007b, 0x0ff6, 0xffa6,
    0xffa7, 0xffa8, 0xffa9, 0xffaa, 0xffab, 0xffac, 0xffad, 0x00fa,
    0x0ff7, 0xffae, 0xffaf, 0xffb0, 0xffb1, 0xffb2, 0xffb3, 0xffb4,
    0xffb5, 0x01f8, 0x7fc0, 0xffb6, 0xffb7, 0xffb8, 0xffb9, 0xffba,
    0xffbb, 0xffbc, 0xffbd, 0x01f9, 0xffbe, 0xffbf, 0xffc0, 0xffc1,
    0xffc2, 0xffc3, 0xffc4, 0xffc5, 0xffc6, 0x01fa, 0xffc7, 0xffc8,
    0xffc9, 0xffca, 0xffcb, 0xffcc, 0xffcd, 0xffce, 0xffcf, 0x03f9,
    0xffd0, 0xffd1, 0xffd2, 0xffd3, 0xffd4, 0xffd5, 0xffd6, 0xffd7,
    0xffd8, 0x03fa, 0xffd9, 0xffda, 0xffdb, 0xffdc, 0xffdd, 0xffde,
    0xffdf, 0xffe0, 0xffe1, 0x07f8, 0xffe2, 0xffe3, 0xffe4, 0xffe5,
    0xffe6, 0xffe7, 0xffe8, 0xffe9, 0xffea, 0xffeb, 0xffec, 0xffed,
```

```
    0xffee,0xffef,0xfff0,0xfff1,0xfff2,0xfff3,0xfff4,0x07f9,
    0xfff5,0xfff6,0xfff7,0xfff8,0xfff9,0xfffa,0xfffb,0xfffc,
    0xfffd,0xfffe
    };

//色差 AC
unsigned char  ht3[] = {
    0x00,0x02,0x01,0x02,0x04,0x04,0x03,0x04,
    0x07,0x05,0x04,0x04,0x00,0x01,0x02,0x77,
    0x00,0x01,0x02,0x03,0x11,0x04,0x05,0x21,
    0x31,0x06,0x12,0x41,0x51,0x07,0x61,0x71,
    0x13,0x22,0x32,0x81,0x08,0x14,0x42,0x91,
    0xA1,0xB1,0xC1,0x09,0x23,0x33,0x52,0xF0,
    0x15,0x62,0x72,0xD1,0x0A,0x16,0x24,0x34,
    0xE1,0x25,0xF1,0x17,0x18,0x19,0x1A,0x26,
    0x27,0x28,0x29,0x2A,0x35,0x36,0x37,0x38,
    0x39,0x3A,0x43,0x44,0x45,0x46,0x47,0x48,
    0x49,0x4A,0x53,0x54,0x55,0x56,0x57,0x58,
    0x59,0x5A,0x63,0x64,0x65,0x66,0x67,0x68,
    0x69,0x6A,0x73,0x74,0x75,0x76,0x77,0x78,
    0x79,0x7A,0x82,0x83,0x84,0x85,0x86,0x87,
    0x88,0x89,0x8A,0x92,0x93,0x94,0x95,0x96,
    0x97,0x98,0x99,0x9A,0xA2,0xA3,0xA4,0xA5,
    0xA6,0xA7,0xA8,0xA9,0xAA,0xB2,0xB3,0xB4,
    0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xC2,0xC3,
    0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xD2,
    0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,
    0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,
    0xEA,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,
    0xF9,0xFA
    };
unsigned char  hsizeT3[] = {
     2, 2, 3, 4, 5, 5, 6, 7, 9,10,12, 4, 6, 8, 9,11,
    12,16,16,16,16, 5, 8,10,12,15,16,16,16,16,16, 5,
     8,10,12,16,16,16,16,16,16, 6, 9,16,16,16,16,16,
    16,16,16, 6,10,16,16,16,16,16,16,16,16, 7,11,16,
    16,16,16,16,16,16,16, 7,11,16,16,16,16,16,16,16,
    16, 8,16,16,16,16,16,16,16,16,16, 9,16,16,16,16,
    16,16,16,16,16, 9,16,16,16,16,16,16,16,16,16, 9,
    16,16,16,16,16,16,16,16,16, 9,16,16,16,16,16,16,
    16,16,16,11,16,16,16,16,16,16,16,16,16,14,16,16,
    16,16,16,16,16,16,16,10,15,16,16,16,16,16,16,16,
    16,16
    };
int hcodeT3[] = {
    0x0000,0x0001,0x0004,0x000a,0x0018,0x0019,0x0038,0x0078,
    0x01f4,0x03f6,0x0ff4,0x000b,0x0039,0x00f6,0x01f5,0x07f6,
    0x0ff5,0xff88,0xff89,0xff8a,0xff8b,0x001a,0x00f7,0x03f7,
```

```
    0x0ff6,0x7fc2,0xff8c,0xff8d,0xff8e,0xff8f,0xff90,0x001b,
    0x00f8,0x03f8,0x0ff7,0xff91,0xff92,0xff93,0xff94,0xff95,
    0xff96,0x003a,0x01f6,0xff97,0xff98,0xff99,0xff9a,0xff9b,
    0xff9c,0xff9d,0xff9e,0x003b,0x03f9,0xff9f,0xffa0,0xffa1,
    0xffa2,0xffa3,0xffa4,0xffa5,0xffa6,0x0079,0x07f7,0xffa7,
    0xffa8,0xffa9,0xffaa,0xffab,0xffac,0xffad,0xffae,0x007a,
    0x07f8,0xffaf,0xffb0,0xffb1,0xffb2,0xffb3,0xffb4,0xffb5,
    0xffb6,0x00f9,0xffb7,0xffb8,0xffb9,0xffba,0xffbb,0xffbc,
    0xffbd,0xffbe,0xffbf,0x01f7,0xffc0,0xffc1,0xffc2,0xffc3,
    0xffc4,0xffc5,0xffc6,0xffc7,0xffc8,0x01f8,0xffc9,0xffca,
    0xffcb,0xffcc,0xffcd,0xffce,0xffcf,0xffd0,0xffd1,0x01f9,
    0xffd2,0xffd3,0xffd4,0xffd5,0xffd6,0xffd7,0xffd8,0xffd9,
    0xffda,0x01fa,0xffdb,0xffdc,0xffdd,0xffde,0xffdf,0xffe0,
    0xffe1,0xffe2,0xffe3,0x07f9,0xffe4,0xffe5,0xffe6,0xffe7,
    0xffe8,0xffe9,0xffea,0xffeb,0xffec,0x3fe0,0xffed,0xffee,
    0xffef,0xfff0,0xfff1,0xfff2,0xfff3,0xfff4,0xfff5,0x03fa,
    0x7fc3,0xfff6,0xfff7,0xfff8,0xfff9,0xfffa,0xfffb,0xfffc,
    0xfffd,0xfffe
    };

typedef struct {
    // SOF
    int width;
    int height;

    // MCU
    int mcu_h[3];
    int mcu_v[3];
    int mcu_width;
    int mcu_height;
    int max_h,max_v;

    int mcu_buf[32*32*3]; //バッファ
    int mcu_preDC[3];

    // DQT
    int dqt[2][64];
    int n_dqt;



    // FILE i/o
    FILE *fp;
    unsigned long bit_buff;
    int bit_remain;

}JPEG;

// ------------------------ I/O ---------------------------
```

```
void put_word(JPEG *jpeg,int v)
{
    unsigned char h,l;
    h = (v>>8)&0xFF;
    l = v&0xFF;

    fwrite(&h,1,1,jpeg->fp);
    fwrite(&l,1,1,jpeg->fp);
}
void put_byte(JPEG *jpeg,unsigned char v)
{
    fwrite(&v,1,1,jpeg->fp);
}
void put_byte_vector(JPEG *jpeg,unsigned char v)
{
  local_b[K]=v;
  K++;
}

// vの下位bitsビットを出力
void put_bits(JPEG *jpeg,unsigned int v,int bits)
{
    unsigned char c;
    int buff,remain;

    buff = jpeg->bit_buff;
    remain=jpeg->bit_remain;

    //バッファに追加
    buff   = (buff << bits) | (v & ((1<<bits)-1));
    remain = remain + bits;

    //printf("%d:%04x¥n",remain,buff);

    //フラッシュアウト
    while(remain > 8) {
        // 1234 5678 9012 3456
        //    ^remain
        //              ^remain-8
        c = (buff >> (remain-8))&0xFF;
       //put_byte(jpeg,c);
           put_byte_vector(jpeg,c);
        //エスケープ

           if(c == 0xFF){
           //put_byte(jpeg,0x00);
              put_byte_vector(jpeg,0x00);
```

```
            }
        remain -= 8;
        //printf("%d:%04x¥n",remain,buff);
    }
    jpeg->bit_buff = buff;
    jpeg->bit_remain=remain;
}

// --------------- インターフェイス実装 ----------------------


// (x0,y0)-(x1,y1)->(8x8)
//最近傍のみでがんばる
void jpeg_encode_bitblt(int *dest,int *src,int width,
                        int x0,int y0,int x1,int y1)
{
    int x,y;
    int u,v; // 元画像での座標
   // printf("(%d,%d)-(%d,%d)¥n",x0,y0,x1,y1);
    for(y=0;y<8;y++){
        v = y0 + (y1-y0) * y / 8;
        for(x=0;x<8;x++){
            u = x0 + (x1-x0) * x / 8;
            dest[x|(y<<3)] = src[width*v + u];
        }
    }
}

#define MY_ABS(X) ((X)>0 ? (X):-(X))

void jpeg_encode_huffman(JPEG *jpeg,int *block,int scan,
                         unsigned char *dc_size,int *dc_code,
                         unsigned char *ac_size,int *ac_code)
{
    int i,run,code;
    int val,val_abs,bitcount;

    // DC
    val = block[0] - jpeg->mcu_preDC[scan];
    jpeg->mcu_preDC[scan] = block[0];

    val_abs = MY_ABS(val);
    //ビット数の計算
    bitcount = 0;
    while(val_abs > 0) {
        val_abs = val_abs >> 1;
        bitcount++;
    }
```

```
    //printf("DC:%d:%d bits¥n",val,bitcount);
    put_bits(jpeg,dc_code[bitcount],dc_size[bitcount]);
    if(bitcount>0) {
        if(val<0) val--;
        put_bits(jpeg,val,bitcount);
    }

    // AC
    run = 0;
    for(i=1;i<64;i++) {
        val = block[i];
        val_abs = MY_ABS(val); //絶対値
        if(val_abs != 0){
            while(run > 15){
                //ZRL
                //printf("ZRL¥n");
                            put_bits(jpeg,ac_code[151],ac_size[151]);
                run -= 16;
            }
            bitcount = 0;
            while(val_abs > 0) {
                val_abs = val_abs >> 1;
                bitcount++;
            }
            code = run * 10 + bitcount + ((run == 15) ? 1 : 0);
            //printf("code:%d(run %d,bitcount %d)¥n",code,run,bitcount);
               put_bits(jpeg,ac_code[code],ac_size[code]);//
            if(val < 0)
                val--;
                    put_bits(jpeg,val,bitcount);
            run = 0;
        }
        else
        {
                    if(i==63)
                            put_bits(jpeg,ac_code[0],ac_size[0]); // EOB
            else
                run++;
        }
  }
}

void jpeg_encode_mcu(JPEG *jpeg)
{
    int scan;
    int h,v,i;
    int hh,vv;
    int *p,*qt;
```

```
unsigned char *dc_size,*ac_size;
int *dc_code,*ac_code;
int block[64],dest[64];
//ジグザグテーブル
static int zigzag[]={
     0,  1,  8,  16,9,  2,  3,10,
    17,24,32,25,18,11,  4,  5,
    12,19,26,33,40,48,41,34,
    27,20,13,  6,  7,14,21,28,
    35,42,49,56,57,50,43,36,
    29,22,15,23,30,37,44,51,
    58,59,52,45,38,31,39,46,
    53,60,61,54,47,55,62,63
    };

// scan:3 固定
for(scan=0;scan<3;scan++)
{
    hh = jpeg->mcu_h[scan];
    vv = jpeg->mcu_v[scan];
    if(scan==0){
        qt = jpeg->dqt[0];
        dc_size = hsizeT0;
        dc_code = hcodeT0;
        ac_size = hsizeT1;
        ac_code = hcodeT1;
    }
    else{
        qt = jpeg->dqt[1];
        dc_size = hsizeT2;
        dc_code = hcodeT2;
        ac_size = hsizeT3;
        ac_code = hcodeT3;
    }
    //printf("scan %d,%dx%d¥n",scan,hh,vv);
    for(v=0;v<vv;v++) {
        for(h=0;h<hh;h++) {
            //MCU からブロックをとってくる
            p = jpeg->mcu_buf + (scan * 1024);
            jpeg_encode_bitblt(block,p,jpeg->mcu_width,
                               jpeg->mcu_width * h / hh,
                               jpeg->mcu_height* v / vv,
                               jpeg->mcu_width * (h+1) / hh,
                               jpeg->mcu_height* (v+1) / vv);

            //jpeg_dct(block,dest);
            ChenDct(block,dest);
```

```
                //zigzag+量子化
                for(i=0;i<64;i++)
                    block[i] = dest[zigzag[i]] / qt[i];

                /*for(i=0;i<64;i++)
                    printf("%03d,",block[i]);
                printf("¥n");*/

                //ハフマン
                jpeg_encode_huffman(jpeg,block,scan,
                                    dc_size,dc_code,
                                    ac_size,ac_code);
            }
        }
    }
}


// RGB->YCbCr
void jpeg_encode_yuv(JPEG *jpeg,int h,int v,unsigned char *rgb)
{
    int Y,U,V;
    int R,G,B;
    int x,y,rx,ry;
    int x0,y0;
    int mw,mh;

    mw = jpeg->mcu_width;
    mh = jpeg->mcu_height;
    x0 = h * mw;
    y0 = v * mh;


    for(y=0;y<mh;y++){
        ry = y + y0;
        if(ry >= jpeg->height)
            ry = jpeg->height-1;

        for(x=0;x<mw;x++){
            rx = x + x0;
            if(rx >= jpeg->width)
                rx = jpeg->width-1;

            R = rgb[ ((ry * jpeg->width) + rx)*3     ] - 0x80;
            G = rgb[ ((ry * jpeg->width) + rx)*3 + 1 ] - 0x80;//RGBに抽出
            B = rgb[ ((ry * jpeg->width) + rx)*3 + 2 ] - 0x80;

            // YCbCr変換
```

```
            Y = (R * 0x4C8 + G * 0x964 + B * 0x1D2)>>12;
            U = (B * 0x800 - R * 0x2B3 - G * 0x54C)>>12; // 順番がズレていることに注意
            V = (R * 0x800 - G * 0x6B2 - B * 0x14D)>>12;

            // クリップ(要らなかった)
            //Y = (Y<0) ? 0 : ((Y>255) ? 255 : Y );
            //U = (U<0) ? 0 : ((U>255) ? 255 : U );
            //V = (V<0) ? 0 : ((V>255) ? 255 : V );
            //printf("(%d,%d,%d)->(%d,%d,%d)¥n",R,G,B,Y,U,V);

                    //printf("%d,%d,%d,%d¥n",mh,mw,y,x);
            jpeg->mcu_buf[(y*mw)+x       ] = Y;
            jpeg->mcu_buf[(y*mw)+x + 1024] = U;
            jpeg->mcu_buf[(y*mw)+x + 2048] = V;
        }
    }
}
int jpeg_hedder(JPEG *jpeg)
{
    int h,v;
    unsigned char sof[] = {
        0xFF,0xC0,//SOF0(フレーム開始：基本 DCT 方式)
        0x00,0x11,//Lf(フレームヘッダ長) = 17 bytes
        0x08,     //標本化精度 = 8 bits(固定)
        0x00,0x00,//走査線本数(Height)
        0x00,0x00,//走査線当たりのサンプル数(Width)
        0x03,     //画像成分数 = 3(フルカラー)
        0x00,0x00,0x00,//輝度成分指定パラメータ 量子化表 = 0
        0x01,0x00,0x01,//色差成分指定パラメータ 量子化表 = 1
        0x02,0x00,0x01 //色差成分指定パラメータ 量子化表 = 1
        };
    unsigned char sos[] = {
        0xFF,0xDA,//SOS
        0x00,0x0C,//Ls
        0x03,     //成分数 = 3(フルカラー)
        0x00,0x00,//輝度ハフマン符号表指定
        0x01,0x11,//色差ハフマン符号表指定
        0x02,0x11,//色差ハフマン符号表指定
        0x00,0x3F,0x00//未使用 (固定)
        };

    // ヘッダ吐き出し

    //SOI
    //printf("SOI¥n");
    put_byte(jpeg,0xFF);
    put_byte(jpeg,0xD8);
```

```
//DQT
//printf("DQT¥n");
put_byte(jpeg,0xFF);
put_byte(jpeg,0xDB);
put_word(jpeg,132); // 2 + (1+64)*2
put_byte(jpeg,0x00); // QT[0]
for(h=0;h<64;h++)
    put_byte(jpeg,jpeg->dqt[0][h]);
put_byte(jpeg,0x01); // QT[1]
for(h=0;h<64;h++)
    put_byte(jpeg,jpeg->dqt[1][h]);

//DHT
//printf("DHT¥n");

put_byte(jpeg,0xFF);
put_byte(jpeg,0xC4);
put_word(jpeg,0x01A2);
put_byte(jpeg,0x00);
for(h=0;h<sizeof(ht0);h++)
    put_byte(jpeg,ht0[h]);
put_byte(jpeg,0x10);
for(h=0;h<sizeof(ht1);h++)
    put_byte(jpeg,ht1[h]);
put_byte(jpeg,0x01);
for(h=0;h<sizeof(ht2);h++)
    put_byte(jpeg,ht2[h]);
put_byte(jpeg,0x11);
for(h=0;h<sizeof(ht3);h++)
    put_byte(jpeg,ht3[h]);

//SOF
//printf("SOF¥n");

sof[5] = (jpeg->height >> 8) & 0xFF;
sof[6] = jpeg->height & 0xFF;
sof[7] = (jpeg->width >> 8) & 0xFF;
sof[8] = jpeg->width & 0xFF;
sof[11] = (jpeg->mcu_h[0] << 4)|jpeg->mcu_v[0];
sof[14] = (jpeg->mcu_h[1] << 4)|jpeg->mcu_v[1];
sof[17] = (jpeg->mcu_h[2] << 4)|jpeg->mcu_v[2];
for(h=0;h<sizeof(sof);h++)
    put_byte(jpeg,sof[h]);

//SOS
//printf("SOS¥n");

for(h=0;h<sizeof(sos);h++)
```

```
        put_byte(jpeg,sos[h]);

  return 0;
}
int jpeg_encode(JPEG *jpeg,unsigned char *rgb){
  int h_unit;
  int v_unit;
  int h,v;

  jpeg->mcu_width  = 8 * jpeg->max_h;
    jpeg->mcu_height = 8 * jpeg->max_v;
    jpeg->mcu_preDC[0]=0;
    jpeg->mcu_preDC[1]=0;
    jpeg->mcu_preDC[2]=0;

    h_unit = jpeg->width  / jpeg->mcu_width;
    if((jpeg->width % jpeg->mcu_width) > 0)
        h_unit++;

    v_unit =( (jpeg->height) / jpeg->mcu_height);
    if((jpeg->height % jpeg->mcu_height) > 0)
        v_unit++;

    //printf("h_unit %d¥n",h_unit);
    //printf("v_unit %d¥n",v_unit);

   // printf("encode¥n");
    for(v=0;v<v_unit;v++) {
        for(h=0;h<h_unit;h++) {
            //MCU を作る(rgb->YCbCr)
            jpeg_encode_yuv(jpeg,h,v,rgb);       //＜＜----rgb 使用

            //MCU をエンコードする(YCbCr->8x8->DCT)
           jpeg_encode_mcu(jpeg);
        }
    }
   return 0;
}
// サンプリングファクタ設定
void jpeg_sample_set(JPEG *jpeg,int compo,int h,int v)
{
    //printf("sample_set(%d,%d,%d)¥n",compo,h,v);
    jpeg->mcu_h[compo] = h;
    jpeg->mcu_v[compo] = v;
    if(jpeg->max_h < h)
        jpeg->max_h = h;
    if(jpeg->max_v < v)
        jpeg->max_v = v;
```

```
}

// 量子化テーブル設定
void jpeg_qt_set(JPEG *jpeg,int *qt0,int *qt1)
{
    int i;
    for(i=0;i<64;i++){
        jpeg->dqt[0][i] = qt0[i];
        jpeg->dqt[1][i] = qt1[i];
    }
}

// エンコードの設定
int jpeg_encode_init(JPEG *jpeg,int width,int height,int quality)
{
    int i,v;

    //量子化テーブル設定
    if(quality < 50){
        for(i=0;i<64;i++)
            jpeg->dqt[0][i] = jpeg_qt[i] * 50 / quality;
        for(i=0;i<64;i++)
            jpeg->dqt[1][i] = jpeg_qt[64+i] * 50 / quality;
    }
    else{
        for(i=0;i<64;i++){
            v = jpeg_qt[i] * (101-quality) / 50;
            if(v == 0) v = 1;
            jpeg->dqt[0][i] = v;
        }
        for(i=0;i<64;i++){
            v = jpeg_qt[i+64] * (101-quality) / 50;
            if(v == 0) v = 1;
            jpeg->dqt[1][i] = v;
        }
    }

    // サイズ設定
    jpeg->width = width;
    jpeg->height= height;

    //サンプリングファクタ設定 (1:1:1) (最大サンプリングファクタの関係で.)
    jpeg_sample_set(jpeg,0,1,1);
    jpeg_sample_set(jpeg,1,1,1);
    jpeg_sample_set(jpeg,2,1,1);

    jpeg->bit_buff=0;
    jpeg->bit_remain=0;
```

```
    return 0;
}

extern int bmp_read(char *,int *,int *,char **);
int main(int argc,char *argv[])
{
    JPEG jpeg;
    FILE *fp;
    char *rgb,fn[256];
    int width,height;
    int i,quality;
    int scan,u,v;


  /*MPI で使用する変数*/
  int proc,cp,tag_pak,tag_num,num1,num2,s;
  char *local_a,*local_c;
  char *m,*r;
  char *read;
  char *write;
  int size;
  char ff[40];//ファイル名
  double s_time,e_time,ss_time,ee_time,st;
  double sread_time,eread_time,read_time;
  double swait_time,ewait_time,wait_time;
  double swrite_time,ewrite_time,write_time;
  double wait_timer[15],read_timer[15];
  double all_timer[15],keisoku_timer[15];
  double write_timer[15];
  int ii,iii;
  int h,w;

  MPI_Status stat;
  MPI_Request request_pak;
  MPI_Request request_num;

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);
  MPI_Comm_size(MPI_COMM_WORLD,&proc);

  s=0;
  st=0;
  ii=0;
  read_time=0;
  wait_time=0;
  write_time=0;
  if(myid==0)
                    s_time=MPI_Wtime();
```

```
while(s<N){
                    if(myid==0){
                              printf("%d¥n",s);
                              sprintf(ff,"//ishi-ken1/mpi/bmp/%d.bmp",s);//
                              sread_time=MPI_Wtime();
                              bmp_read(ff,&width,&height,&rgb);
                              eread_time=MPI_Wtime();
                              read_time += eread_time-sread_time;
                              //bmp_read("//ishi-ken1/mpi/bmp/.bmp",&width,&height,&rgb);//argv[1]

                    }
                    MPI_Bcast(&width,1,MPI_INT,0,MPI_COMM_WORLD);
                    MPI_Bcast(&height,1,MPI_INT,0,MPI_COMM_WORLD);

                    if(myid!=0){
                              rgb=(char *)malloc(width * height * 3 + 256);
                    }


                    local_a = (char *)malloc(width * height * 3 + 256);
                    local_b = (char *)malloc(width * height * 3 + 256);
                    local_c = (char *)malloc(width * height * 3 + 256);
                    read    = (char *)malloc(width * height * 3 + 256);
                    write   = (char *)malloc(width * height * 3 + 256);
                    r       = (char *)malloc(width * height * 3 + 256);

                    if(rgb == NULL || width == 0)
                              return 0;

                    if(myid==0)
                              swait_time=MPI_Wtime();


MPI_Scatter(rgb,(width*height*3)/proc,MPI_CHAR,local_a,(width*height*3)/proc,MPI_CHAR,0,MPI_COMM_WORLD);

                    if(myid==0){
                              ewait_time=MPI_Wtime();
                              wait_time+=ewait_time - swait_time;
                    }

          tag_pak=1;
          tag_num=2;

          cp=0;
          i=0;
                    /*if(myid==0)
                              s_time=MPI_Wtime();//計算時間のみの計測を開始
*/
```

```
sprintf(ff,"//ishi-ken1/mpi/jpeg/file_%d_rank_%d.jpg",s,myid);
        if(myid!=0){
                        fp = fopen(ff,"wb");
                        jpeg.fp=fp;
                jpeg_encode_init(&jpeg,width,height/proc,qua);
                        jpeg_hedder(&jpeg);
                jpeg_encode(&jpeg,local_a);
                        //printf("cpu No %d  =  %d",myid,K);
                        //MPI_Send(&K,1,MPI_INT,0,tag_num,MPI_COMM_WORLD);
                        //MPI_Send(local_b,K,MPI_CHAR,0,tag_pak,MPI_COMM_WORLD); //ホストに送る

                        //MPI_Wait(&request_pak,&stat);
        }

        else if(myid==0){
                ss_time=MPI_Wtime();
                        fp = fopen(ff,"wb");
                        jpeg.fp=fp;
                        jpeg_encode_init(&jpeg,width,height/proc,qua);
                        jpeg_hedder(&jpeg);

                jpeg_encode_init(&jpeg,width,height/proc,qua);
                jpeg_encode(&jpeg,local_a);
        ee_time=MPI_Wtime();
        st +=ee_time-ss_time;
        }
        if(myid==0){
                swrite_time=MPI_Wtime();
        }
        for(iii=0;iii<=K;iii++){
                fwrite(&local_b[iii],1,1,jpeg.fp);
        }
        MPI_Barrier(MPI_COMM_WORLD);
        if(myid==0){
                ewrite_time=MPI_Wtime();
                write_time+=ewrite_time - swrite_time;
        }
                        //EOI
put_byte(&jpeg,0xFF);
put_byte(&jpeg,0xD9);
fclose(jpeg.fp);

free(rgb);
free(local_a);
free(local_b);
free(local_c);
free(read);
```

```
free(write);

if(myid==0){
        if(s==0){
                e_time = MPI_Wtime();
                all_timer[ii]=e_time - s_time;
                wait_timer[ii]=wait_time;
                read_timer[ii]=read_time;
                write_timer[ii]=write_time;
                keisoku_timer[ii]=st;
                ii++;
        }
        if(s==4){
                e_time = MPI_Wtime();
                all_timer[ii]=e_time - s_time;
                wait_timer[ii]=wait_time;
                read_timer[ii]=read_time;
                write_timer[ii]=write_time;
                keisoku_timer[ii]=st;
                ii++;
        }
        if(s==9){
                e_time = MPI_Wtime();
                all_timer[ii]=e_time - s_time;
                wait_timer[ii]=wait_time;
                read_timer[ii]=read_time;
                write_timer[ii]=write_time;
                keisoku_timer[ii]=st;
                ii++;
        }
        if(s==24){
                e_time = MPI_Wtime();
                all_timer[ii]=e_time - s_time;
                wait_timer[ii]=wait_time;
                read_timer[ii]=read_time;
                write_timer[ii]=write_time;
                keisoku_timer[ii]=st;
                ii++;
        }
        if(s==49){
                e_time = MPI_Wtime();
                all_timer[ii]=e_time - s_time;
                wait_timer[ii]=wait_time;
                read_timer[ii]=read_time;
                write_timer[ii]=write_time;
                keisoku_timer[ii]=st;
                ii++;
        }
```

```
                if(s==99){
                        e_time = MPI_Wtime();
                        all_timer[ii]=e_time - s_time;
                        wait_timer[ii]=wait_time;
                        read_timer[ii]=read_time;
                        write_timer[ii]=write_time;
                        keisoku_timer[ii]=st;
                        ii++;
                }

        }
        s++;
        K=0;
        }

        MPI_Barrier(MPI_COMM_WORLD);//計測時間のために一度同期を取る
        if(myid==0){
                e_time = MPI_Wtime();//計測の終了
                for(i=0;i<5;i++){
                        printf("case %d ¥n",i);
                        printf("処理時間        %10.6f¥n",all_timer[i]);
                        printf("読み込み時間    %10.6f¥n",read_timer[i]);
                        printf("書き込み時間    %10.6f¥n",write_timer[i]);
                        printf("送受信時間      %10.6f¥n",wait_timer[i]);
                        printf("計算時間        %10.6f¥n",keisoku_timer[i]);
                }
                printf("case 5 ¥n");
                printf("処理時間        %10.6f¥n",e_time - s_time);
                printf("読み込み時間    %10.6f¥n",read_timer[5]);
                printf("書き込み時間    %10.6f¥n",write_timer[i]);
                printf("送受信時間      %10.6f¥n",wait_timer[5]);
                printf("計算時間        %10.6f¥n",st);
        }




  MPI_Finalize();



    return 0;

}
```

bmp.h

```
#ifndef __BMP_H
#define __BMP_H
int bmp_rgb2bgr(int width,int height,unsigned char *rgb);
int bmp_write(char *name,int width,int height,unsigned char *image);
#endif
```

bmp.c

```
//BMP ローダー

#include <stdio.h>

typedef struct
{
    unsigned char  id[2];
    unsigned long  filesize;
    unsigned short reserve1;
    unsigned short reserve2;
    unsigned long  offset; // 26:12,54:40,122:108
    unsigned long  headsize; // 12,40,108
    unsigned long  width;
    unsigned long  height;
    unsigned short plane;
    unsigned short bpp;
    unsigned long  method;
    unsigned long  datasize;
    unsigned long  x_dpm;
    unsigned long  y_dpm;
    unsigned long  palnum;
    unsigned long  imp_pal;
}BMP_header;

// 24bit RGB 限定.
int bmp_read(char *name,int *width,int *height,char **rgb)
{
    BMP_header bh;
    FILE *fp;
    int x,y,k;
    int r,g,b;
    int mod;
    char *p;

  printf("BMP READ¥n");
  *width = *height=0;
    fp = fopen(name,"rb");
    if(!fp)
           return -1;
    fread( bh.id, 1,2 ,fp);
    if(bh.id[0] != 'B' || bh.id[1] != 'M'){
```

```
            printf("%c%c¥n",bh.id[0],bh.id[1]);
            return 0;
    }
    fread( &(bh.filesize), 4,1,fp);
    fread( &(bh.reserve1), 2,1,fp);
    fread( &(bh.reserve2), 2,1,fp);
    fread( &(bh.offset  ), 4,1,fp);
    fread( &(bh.headsize), 4,1,fp);
    fread( &(bh.width   ), 4,1,fp);
    fread( &(bh.height  ), 4,1,fp);
    fread( &(bh.plane   ), 2,1,fp);
    fread( &(bh.bpp     ), 2,1,fp);
    fread( &(bh.method  ), 4,1,fp);
    fread( &(bh.datasize), 4,1,fp);
    fread( &(bh.x_dpm   ), 4,1,fp);
    fread( &(bh.y_dpm   ), 4,1,fp);
    fread( &(bh.palnum  ), 4,1,fp);
    fread( &(bh.imp_pal ), 4,1,fp);
  printf("%dx%d¥n",bh.width,bh.height);
  printf("bpp %d¥n",bh.bpp);
  if(bh.bpp != 24){
            return 0;
    }
  *width = bh.width;
  *height= bh.height;
  *rgb = p = (char *)malloc(bh.width * bh.height * 3 + 256);
    mod = (bh.width*3)%4;
    if(mod)
        mod = 4-mod;
    for(y=0;y<bh.height;y++){
        for(x=0;x<bh.width;x++){
            k = ((bh.height-y-1)*bh.width + x)*3;

            p[k+2] = fgetc(fp); // R
            p[k+1] = fgetc(fp); // G
            p[k  ] = fgetc(fp); // B
        }
        if(mod>0){
            for(x=0;x<mod;x++){
                                fgetc(fp); // 読み捨てる
            }
        }
    }
    fclose(fp);
    return 0;
}
int bmp_write(char *name,int width,int height,unsigned char *rgb)
{
```

```
BMP_header bh;
FILE *fp;
int x,y,k;
int r,g,b;
int mod;

bh.id[0] = 'B';
bh.id[1] = 'M';
bh.filesize = 54 + (width*height*3);
bh.offset   = 54;
bh.headsize = 40;
bh.width    = width;
bh.height   = height;
bh.plane    = 1;
bh.bpp      = 24;
bh.method   = 0;
bh.datasize = 0;
bh.x_dpm    = 3779;
bh.y_dpm    = 3779;
bh.palnum   = 0;
bh.imp_pal  = 0;

fp = fopen(name,"wb");
if(!fp) return -1;
fwrite( bh.id, 1,2 ,fp);
fwrite( &(bh.filesize), 4,1,fp);
fwrite( &(bh.reserve1), 2,1,fp);
fwrite( &(bh.reserve2), 2,1,fp);
fwrite( &(bh.offset  ), 4,1,fp);
fwrite( &(bh.headsize), 4,1,fp);
fwrite( &(bh.width   ), 4,1,fp);
fwrite( &(bh.height  ), 4,1,fp);
fwrite( &(bh.plane   ), 2,1,fp);
fwrite( &(bh.bpp     ), 2,1,fp);
fwrite( &(bh.method  ), 4,1,fp);
fwrite( &(bh.datasize), 4,1,fp);
fwrite( &(bh.x_dpm   ), 4,1,fp);
fwrite( &(bh.y_dpm   ), 4,1,fp);
fwrite( &(bh.palnum  ), 4,1,fp);
fwrite( &(bh.imp_pal ), 4,1,fp);

mod = (width*3)%4;
if(mod)
    mod = 4-mod;
//printf("mod:%d¥n",mod);
for(y=0;y<height;y++){
    for(x=0;x<width;x++){
        k = ((height-y-1)*width + x)*3;
```

```
            r = rgb[k  ]; // R
            g = rgb[k+1]; // G
            b = rgb[k+2]; // B
            fputc(b,fp);
            fputc(g,fp);
            fputc(r,fp);
        }
        if(mod>0){
            for(x=0;x<mod;x++){
                fputc(0,fp);
            }
        }
    }
    fclose(fp);
    return 0;
}
```