卒業研究報告書

題目

MPI による JPEG 圧縮の検証

指 導 教 員 石水 隆 助手

報告者

03-1-47-073

横瀬拓也

近畿大学理工学部情報学科

平成 19 年 2 月 10 日提出

概要

計算機には、常に高速な処理が求められている。高速処理の手段として、1 台の計算機の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 台の処理速度の向上には限界があり、また、性能の向上には時間や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。だが高速処理が可能となる並列計算機はとても高価なものである。そのため、ネットワークを使用することで、複数の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)を用いてその性能を実験的に評価する。評価方法として、無圧縮 BMP(Bit Map)画像から JPEG(Joint Photographic Experts)画像に変換処理を行なう並列エンコーダをプログラミングし、どれだけの処理時間の短縮が出来ているのか検証を行なう。

目次

1 序論 1

1.1 本研究の背景 1
1.1.1 並列処理 1
1.1.2 並列計算機 1
1.1.3 PVM 1
1.1.4 MPI 2
1.1.5 PVM と MPI の違い 2
1.2 本研究の目的 2
1.3 本研究の構成 2

2 研究内容 3

2.1 準備 3
2.1.1 使用機器 3
2.1.2 MPICH2 の導入 4
2.1.3 使用画像 4
2.1.4 JPEG 4
2.1.5 JPEG エンコード 5
2.2 JPEG 並列エンコーダ 6

3 結果 7

3.1 全体での処理時間 7
3.2 演算時間 8

4 考察 9

5 結論・今後の課題 10

付録 12

1 序論

1.1 本研究の背景

1.1.1 並列処理

ある 1 つの処理を、複数のプロセッサを用い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。

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計算に逐次処理では膨大な時間がかかる場所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

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、全てのプロセッサが同じメモリを通して使用する共有メモリ型並列処理や、それぞれのプロセッサは個々に局所メモリを持ち、通信にはネットワークを使用する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。その中で、共有メモリ型並列処理は同期問題やデータの送受信といった問題がメモリを共有しているため対処しやすいが、プロセッサの増減などの変化があった場合には、メモリに全てのプロセッサを接続させることが困難になってしまう。そのため、現在では局所メモリが使用できる分散メモリ型並列処理が主流となっている。

本来、並列計算機自体はとても高価なものである。そのため、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。

現在は、複数の計算機をネットワークで接続し、仮想的な計算機である仮想計算機(Virtual Machine)を構築するソフトウェアが無料で提供されていることもあり並列計算機は身近なものになっている。また、ソフトウェアの種類には、共有メモリ型並列処理用として OpenMP というものがある、だが共有メモリ型並列処理より分散メモリ型並列処理が利用されているためソフトウェアも MPI・PVM といったものが主流となっている。これらはクラスタ(Cluster)処理やグリッド(Grid)処理にも幅広く使用されている。

MPI や PVM については次に説明したい。

1.1.3 PVM

PVM(Parallel Virtual Machine)[1]は、1991 年に米国のオークリッジ国立研究所(Oak Ride National Laboratory)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、PVM インターフェイスルーチンライブラリである、ライブラリには、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 の大きな特徴として、耐故障性(Fault Tolerant)があげられる、PVM は任意で計算機の追加や削除が行なえる。通常、仮想並列計算機で計算中に、ある 1 台が停止してしまうと、計算処理が出来なくなってしまうが、PVM は故障した計算機を仮想並列計算機内から迅速に削除され、計算処理自体が停止してしまうことがなく続行できる。PVM の問題点として、PVM は多くの並列計算機に移植されるようになった、そのために、各並列計算機ベンダが独自にチューニングを行なった PVM を開発してしまい。PVM で作成をしたプログラムの移植性が乏しくなってしまった。

1.1.4 MPI

MPI(Message Passing Interface)[2]は、メッセージ通信のプログラムを記述するために広く使われる標準を目指して開発された、メッセージ通信の API 仕様である。そのため、PVM と違いソフトウェアがあるというわけではない。

MPI 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。MPI は PVM よりも後に開発されたため、PVM と同等の機能を持っており、問題点の改善も含まれている。

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(Operating System)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している、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。

1.1.5 PVM と MPI の違い

PVM と MPI の大きな違いとして動的なプロセス管理があったが、最近 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とから、PVM の優位性の 1 つが失われている。しかし、MPI-2 は前の規格である MPI-1 の関数も使用できるが、新規格である MPI-2 の機能を完全にサポートしている実装が無いという問題がある。

PVM の利用時の大きな欠点は移植性に乏しいということがいえ、MPI は PVM と違い世界的な標準が目的とされ開発されたものであるため移植性が高く評価されている。

MPI の欠点は、MPI は高レベルのバッファ操作を可能とするために同一のワークステーション・クラスタを対象としていることである、MPI はこの性質上仮想計算機を構成する全ての計算機が同じ OS で無ければ動作がしないため、異機種間での並列処理ができないことである。その点 PVM は MPI と違い異機種間における処理を目的で作成されているので、異機種の計算機による並列処理に適していると言える。

MPI は同機種における並列処理が望ましいため近くにある同機種の計算機でのクラスタ処理が、PVM はデーモンを使用しての TCP/IP 通信ができ、異機種の計算機を使用できるという特性を持っているため、様々な異機種の計算機がある場所や通信を利用し離れた計算機も使用できるという点からクラスタ処理とグリッド処理にそれぞれ利用できる。

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 は現在でも新規格である MPI-2 の研究開発が盛んに行なわれている。

1.2 本研究の目的

本研究では、複数台の計算機をネットワークにおいて構成する MPI による仮想並列計算機の性能を実験的に評価する。評価方法として 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作成することで、1 台の計算機での逐次処理時におけるエンコードの処理時間が、複数台の計算機を使用しての並列エンコーダにおいてどれだけの処理時間の向上が出来ているかの検証を行なう。

1.3 本研究の構成

本研究の構成は以下の通りである。2 章では、本研究の使用機器および MPI の導入方と JPEG の特性について述べる。3 章では、MPI による JPEG の特性の結果を示し、次の 4 章においてその考察を行う。5 章では結果と今後の課題について述べる。

2　研究内容

2.1 準備

2.1.1 使用機器

まず、研究を行なうにあたって仮想並列計算機を構築するソフトウェアを配布している Web Site から使用するソフトウェアをダウンロードし、研究室にある計算機にインストールと環境設定を行なう。今回の研究で使用する仮想並列計算機を構築するソフトウェアは、MPI を無料のソフトウェアとして実装された MPICH2 というソフトウェアを使用することにする。この MPICH2 を使用する理由は、MPI の特徴である移植性の高さ、MPI の新規格である MPI-2 をある程度サポートしている点や、Windows OS にも導入が可能であり現在でも研究開発が盛んに行なわれているため、MPICH2 をしての本研究を行なう。

MPI による仮想並列計算機を構築する場合、OS を統一する必要がある。Microsoft 社が提供販売を行なっている Windows OS は、現在の一般家庭や学校そして企業などの場所において、他の OS に比べ幅広く使用されている。このため、本研究では Windows OS を用いる。

本研究において使用する計算機のスペックを表 1 に示す。

表 1 本研究で使用する計算機のスペック

	PC1	PC2	PC3	PC4
OS	Microsoft Windows 2000	Microsoft Windows XP	Microsoft Windows 2000	Microsoft Windows 2000
CPU	Intel Celeron 768MHz	Intel Pentium 4 1.9GHz	Intel Pentium 4 1.9GHz	Intel Celeron 768MHz
Memory	512MB	670MB	670MB	512MB

本研究で使用する 4 台の計算機は、ルータ(Router)およびハブ(Hub)による LAN(Local Area Network)を構成している。本研究で用いる LAN の構成図を図 1 に示す。

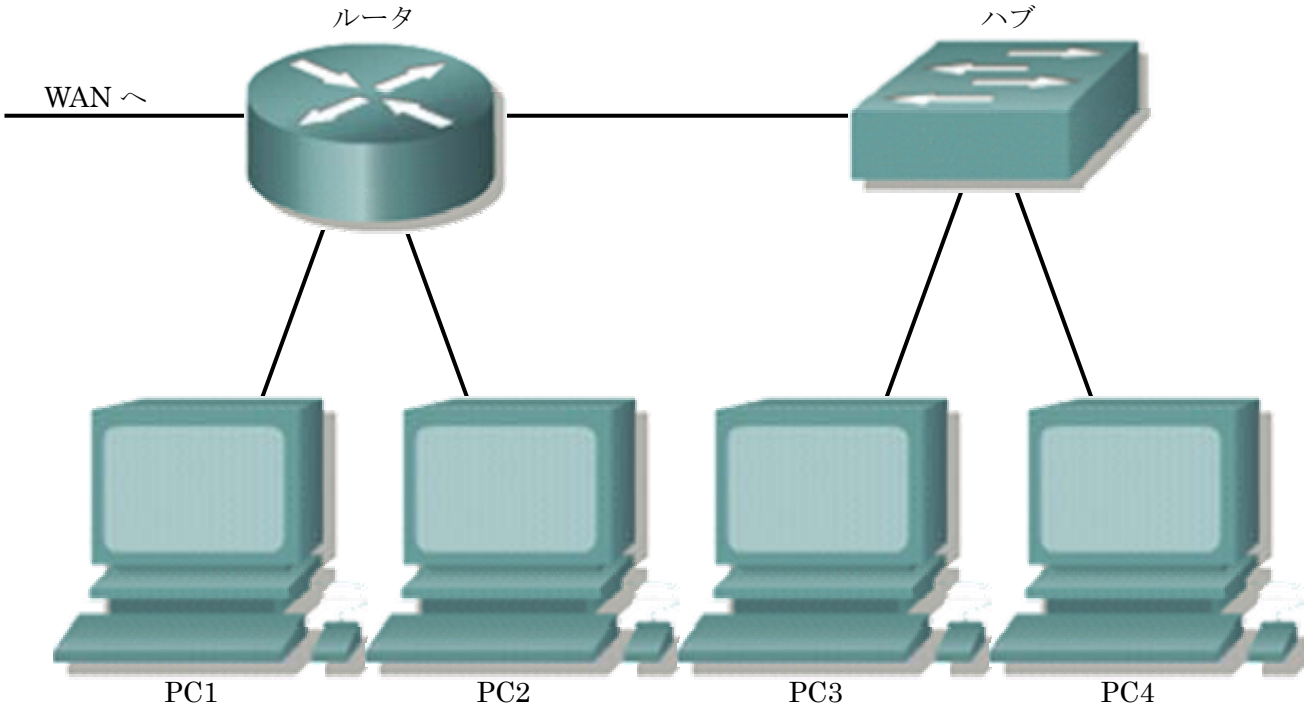


図 1 仮想並列計算機の構成

2.1.2 MPICH2 の導入

MPICH2 を使用するには、計算機に MPICH2 をインストール(Install)する必要がある[4]。インストール自体は Windows 用 MPICH2 の exe ファイルをダウンロードした後に exe ファイルをクリックすることにより、インストーラー(Installer)が起動し自動的にインストールされる。インストールした後に環境設定をしなければいけないため、デフォルト(Default)の C:¥Program Files¥MPICH2 フォルダインストールを行なう。

インストール後の環境設定では、インストール先のフォルダにある bin ファイルに環境変数でパス(PATH)指定をする。

また、Windows OS の場合には全ての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設定しておく必要がある。プロセスの実行には、すべ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、共有するフォルダを作成することで実行ファイルの受け渡しが迅速に行なえるようになる。今回の研究時には、各計算機共通の共有フォルダ上での実行ファイルによる処理において集団通信関数が使用できない不具合が生じたため、各計算機に個別の共有フォルダを作ることで実行ファイル配布時の作業においてなるべく無駄を省くことにした。

今回作成する並列エンコーダに使用するプログラム言語は C 言語で行い、コンパイルツールは Microsoft 社製の Visual C++.net を使用する。プログラミングをするにあたって MPICH2 を使用できるように設定しなければならない。まず、空のプロジェクトを作成し、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を MPICH2 フォルダにある lib と include フォルダを指定し追加を行なう。次にプロジェクトオプション設定でリンカ入力の依存ファイルを追加する、追加する依存ファイルは mpich2.lib である。これらの設定を行なうことで MPIDH2 による並列プログラミングが可能となる。

2.1.3使用画像

本研究で使用する画像は、無圧縮の BMP(Bit Map)画像を使用することにする。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 である。

今回使用する画像の枚数は 100 枚とし。検証の際には、1･5･10･25･50･100 と順に計測を行い各枚数における処理時間を検証する。

2.1.4JPEG

JPEG[5][6]は、”Joint Photographic Experts Group”の頭文字であり、画像データベース・カラーファクシミリ・印刷などの分野において適用するためのカラー静止画像符号化標準化を目指し、ISO(International Organization Standardization) と CCTT(Consultative Committee for International Telegraph and Telephone)の２つの組織がジョイントし設立された検討グループのことである。標準方式の正式名称は”Digital compression and coding of continuous-tone still image”といい、モノクロやカラーの連続的な階調の静止画の符号化技術である。

JPEG は広い適用範囲を想定した汎用性の高いもので、アルゴリズムの機能を４つの動作モードに分類し選択できるようになっている。下の機能紹介に書いてある DCT やブロックについては後ほど説明を行なう。

(1) シーケンシャル DCT ベース:8×8 画素からなるブロックの左から右へ進むライン上の符号化処理を上から下へ順番に 1 回のスキャンで行なう。符号化処理は 2 次元 DCT 係数の量子化と、量子化係数のエントロピー符号化からなる。

(2) プログレッシブ DCT ベース:基本構成はシーケンシャル DCT ベースと同じであるが、処理スキャンの回数が複数回ある。最初のスキャンで大まかな画像が得られ数回行なうことで正確な画像を得られる。

(3) ロスレス(ひずみ無し):DCT 変換を用いずに近接画素との差分をエントロピー符号化するため、符号化におけるひずみが発生しない。

(4) ハイアラーキカル(階層型): 3 つの動作モードを組み合わせることで、複数の空間解像度を持つ画像のピラミッド構造を作成する。

これらの機能を実現する符号化アルゴリズムごとに分別を行なうとDCT方式とSpatial方式も２種類に分けられる。DCT は符号化・複合化によってひずみが生じる非可逆プロセスである。Spatial 方式は符号化に DPCM(Differential PCM)とエントロピー符号化を使用しているため、DCT 方式とのアルゴリズムにおいて連続性がなく、そのため DCT 方式とは異なり符号化・複合化においてのひずみが生じない可逆プロセスである。

現在では、JPEG圧縮に起きるひずみを解消するために、新しいJPEGアルゴリズムであるJPEG2000という規格がでている、JPEG よりも JPEG2000 は高圧縮でひずみも無いという理想的な画像圧縮法であるが、エンコードの処理時間がかかり、また、アルゴリズムも難解なために今でもそれほど広まってはいない。

2.1.5 JPEG エンコード

図 2 のような流れで符号化が行なわれている。この符号化における過程での処理で特に JPEG 処理の特徴である、ブロック分割、DCT 変換、量子化について説明をおこなう。

(1) ブロック分割(標本化):JPEG の DCT 方式では、最初に入力された画像データのサンプリングで、1 枚の画像データを左上端から右端まで直進し、その後下段へ移動し、同様に左端から右端へと進んでいき、8×8 画素から構成されるブロックの集合である MCU(Minimum Coded Unit)と呼ばれる単位に分割する。この分割を行なってできたブロックはブロック１つごとに後の処理が行なわれる。

(2) DCT 変換:DCT 変換(離散コサイン変換、Discrete Cosine Transform)は、複雑な情報をもつ画像データを、複数の単純な空間周波数成分に分解し変換を行なう。この作業によって、飛躍的にデータの削減が可能となる。しかし、この空間周波数成分の係数を実数のままにしておくと、小数点以下のデータを持つことになり、処理効率が非常に悪くなるためハフマン符号化を行なう。

(3) 量子化:データ量の節約のため係数を整数に変換する量子化処理を行なう。量子化処理は、量子化マトリクスに基づき、輝度と色素の１ブロック単位で行なわれる。この作業で共通のデータを増やし容量の削減をする。

次に、量子化されたデータは、出現率の高いデータと出現率の低いデータとに分けることで、長いビットと短いビットに割り当て圧縮していくハフマン符号処理を行なう。

(4) ハフマン符号化:ハフマン符号はＤＣ成分(直流成分)とＡＣ成分(交流成分)とで別々に処理を行なう。ＤＣ成分はブロックの左上端に置かれこれには空間周波数がない。ＡＣ成分はＤＣ成分を取り除いたもので、ジグザクスキャンを行なうことで２次元信号を１次元信号にし、ランレングス方により圧縮をするとともに符号化テーブルを用いて符号化を行なう。

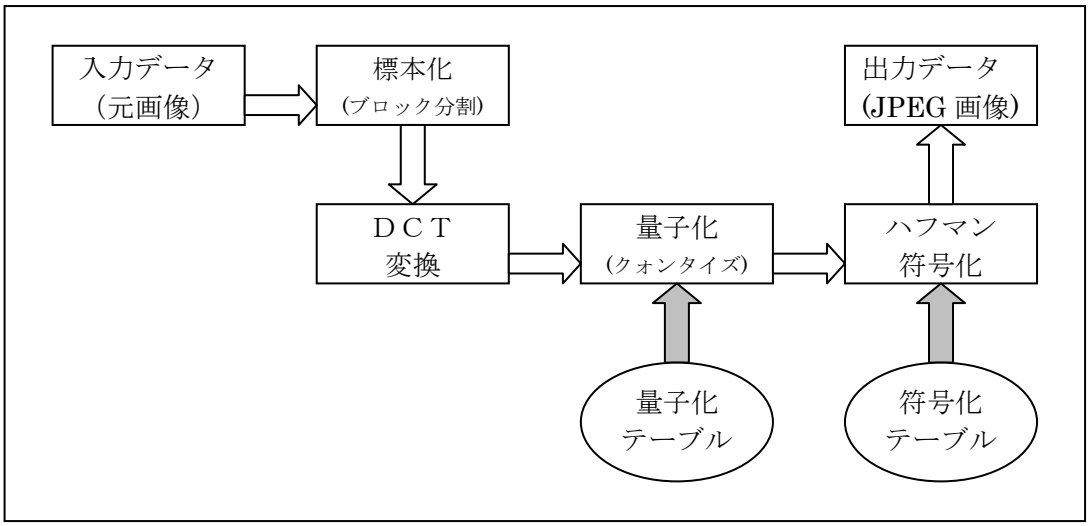


図 2 JPEG エンコーダの流れ

2.2 JPEG 並列エンコーダ

図 3 に JPEG 並列エンコーダの流れ図を示す。JPEG 並列エンコーダは、MPI を起動させ仮想並列計算機を構成している全ての計算機に指示を与える計算機をメイン計算機として処理を行なう。

メイン計算機は、BMP 画像を読み込みその画像を分割する。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。分割を行なう上での注意として、BMP 画像は RGB の 3 色で 1 つの画素を表わしている。データ上ではそれらは 1 次元配列として保存されているため横のサイズは分割時に何の支障も無いが、縦のサイズの場合は台数分で割り切れないことになると画像の分割は出来ない。その際には、割り切れるように計算機の台数を増やすか減らす作業を行なわないといけない。

メイン計算機は、分割を行なった BMP 画像をネットワークでつながっている他のサブ計算機にそれぞれ送信を行なう。サブ計算機はメイン計算機から送信された分割 BMP 画像の受信を行い。メイン計算機とサブ計算機は、各計算機が持っている分割 BMP 画像をそれぞれ JPEG エンコーダによって JPEG データにエンコード処理を行なう。サブ計算機は JPEG データに変換されたデータをメイン計算機に送信をする。メイン計算機は、サブ計算機から送信された JPEG データを受信しファイルに書き込むことで 1 枚の JPEG 画像を出力する。書き込みを行なう際に、送信されてきた JPEG データをそのまま書き込むと画像の順番がおかしくなってしまう恐れがあるため、メイン計算機はサブ計算機に順番を決めることで、分割 BMP 画像を順番どおりにサブ計算機に送信する。これによって書き込みの際にメイン計算機は画像の書き込みの順番を守ることで、正確な 1 枚の JPEG 画像を出力できるようになる。

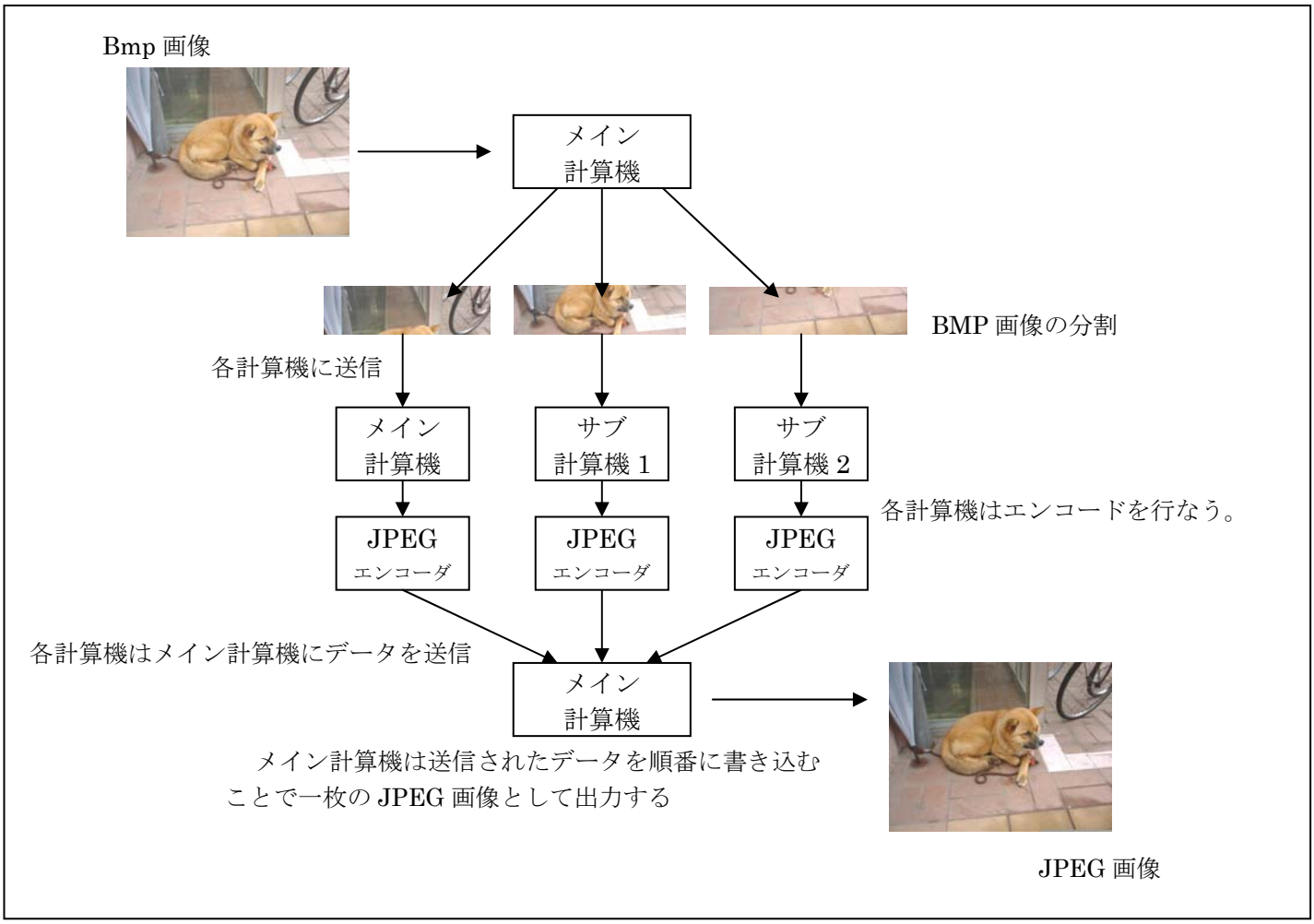


図 3 並列 JPEG エンコーダの流れ

3 結果

今回の実験での検証では、エンコード処理を行なう BMP 画像の枚数を 1･5･10･25･50･100 の順にエンコード処理を行なう、その際に 1 台での処理時間を計測し、次に JPEG 並列エンコーダを構成する計算機の台数を 2 台から 4 台まで 1 台ずつことで、どれだけの処理時間がかかっているかの計測を行なった。

計測に使用する計算機は 2 章において示している。次に仮想並列計算機を構成する場合において使用する組み合わせを下記に示しておく。

(1) CPU 数 1 では PC1 を使用。
(2) CPU 数 2 ではホスト計算機を PC1 に、サブ計算機を PC2 で使用。
(3) CPU 数 3 ではホスト計算機を PC1 に、サブ計算機を PC2･PC3 で使用。
(4) CPU 数 4 ではホスト計算機を PC1 に、サブ計算機を PC2･PC3･PC4 で使用。

3.1 全体での処理時間

ここでは、JPEG 並列エンコーダを実行した際に BMP 画像のデータの読み込みからエンコード処理を行い JPEG 画像として出力されるまでの全体での処理時間の結果を示す。

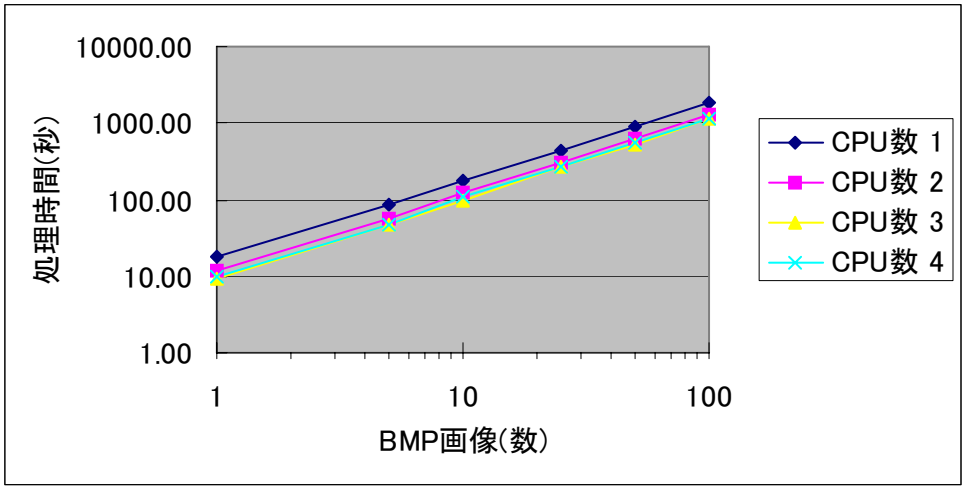
JPEG 並列エンゴーダによる処理時間を表 2 に示す。また、BMP 画像の枚数変化に伴う処理時間の推移をグラフ 1 で表わし、CPU 数の変化に伴う処理時間の推移をグラフ 2 で示す。

表 2 より JPEG 並列エンコーダを使用した場合、全体的に処理時間の向上が行なえていることが分かる。**グラフ 1** より BMP 画像が 1 枚の時から JPEG 並列エンコーダを使用した場合において CPU 数 1 台よりも処理速度の向上が出来ていることが示される。しかし、グラフ 2 より CPU 数 1 から CPU 数 2 においての処理時間が大幅に向上しているが、それ以降 CPU 数を増やしても処理の向上が小さなものになってしまっていることが示され、そればかりか、CPU 数 3 から CPU 数 4 に増やした場合の処理時間が向上するよりも、むしろ悪化している結果となっている。

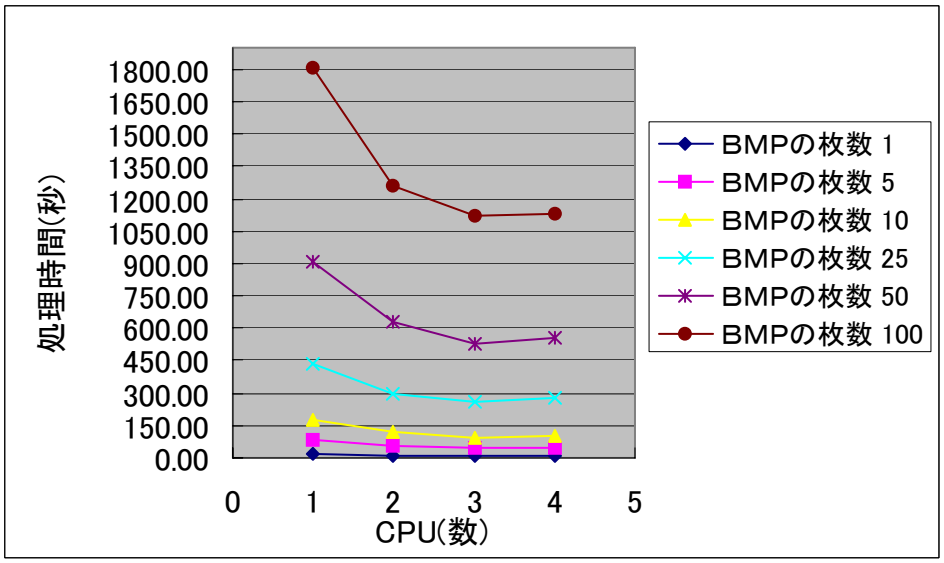
表 2 全体での処理時間

		BMPの枚数					
		1	5	10	25	50	100
CPU数	1	17.74	87.13	174.64	435.89	908.40	1804.00
	2	11.57	56.69	121.72	295.54	631.30	1262.05
	3	9.47	47.92	95.86	262.76	532.01	1118.02
	4	9.90	47.01	106.39	275.50	556.15	1129.84

(秒)



グラフ 1 BMP 画像の枚数変化に伴う全体の処理時間



グラフ 2 CPU 数を変化させた場合における全体の処理時間

3.2 演算時間

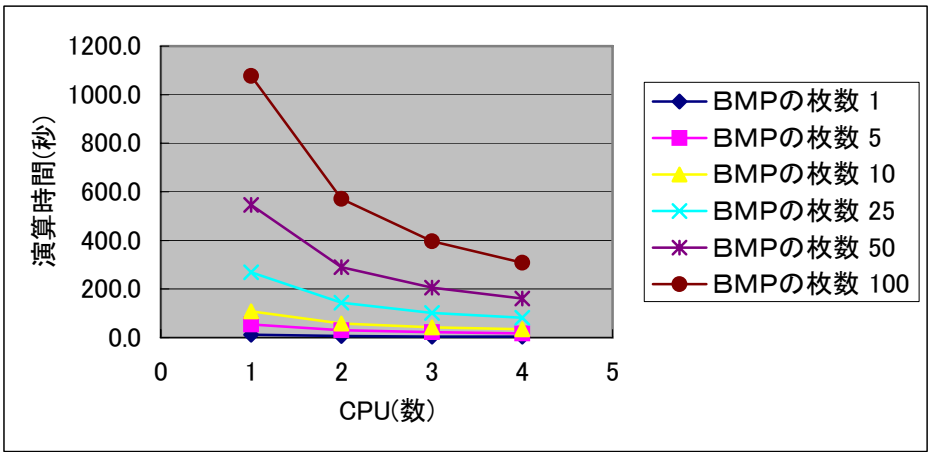
ここでは、BMP 画像データのエンコード処理においての演算時間だけを示す。

表 3 にエンコードにおける演算時間の数値を、グラフ 3 では CPU 数の変化に伴う演算時間の推移を示す。

表 3 およびグラフ 3 より、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ことが示される。

表 3 エンコードにおける演算時間

		BMPの枚数					
		1	5	10	25	50	100
CPU数	1	11.2	54.13	107.59	268.53	546.13	1077.02
	2	6.12	29.89	57.74	143.40	290.45	571.51
	3	4.37	21.89	41.88	101.26	205.67	396.44
	4	3.44	17.65	34.79	81.91	160.84	308.82



グラフ 3 CPU 数を変化させた場合の演算時間

4 考察

今回のこの計測結果では、エンコードだけの演算処理時間を見てみると CPU 数が多くなればなるほど演算時間が短くなる結果になっているが、全体を通しての処理時間を見てみるとあまり短くなっていない。これは、計算機が増えればその分だけ同期時間や通信時間に時間がかかってしまうために起こることだと考えられる。また、計算機のスペックによっては同期時間などに影響が出てくる。この影響は CPU 数 3 と CPU 数 4 での計測結果の処理に現れているように思われる。

CPU 数 3 は低スペックの計算機が 1 台と高スペック計算機が 2 台という構成でメイン計算機を低スペック計算機として処理を行っているが、CPU 数 4 はこの状態に低スペック計算機をサブ計算機に加えて処理を行なっている。そうなると、全体の計算処理時間自体に差が生じてしまう、それが同期時間に時間がかかってしまう原因の 1 つだと考えられる。

5　結論・今後の課題

本研究では、MPICH2 を用いて MPI を結成し、JPEG 並列エンコーダ処理により、その性能を検証した。

本研究の検証により、仮想並列計算機を構成する計算機間のスペック差が小さいときには膨大なデータの処理から少量のデータの処理まで、並列処理が有効であることが示される。一方、仮想並列計算機を構成する計算機の中で低スペック計算機の割合が高い場合では、MPI による並列化はあまり効果的ではない。

MPI 並列計算機は、計算機同士のネットワーク構築時の接続方法をより効率良く行なえる接続にし、また、スペックにあまり違いがない計算機などを使用することで更に速度の向上ができるものと思われる。また、今回の検証で使用した並列エンコーダのアルゴリズムではデータを均等に計算機に送信を行なっていたが、低スペック計算機と高スペック計算機とでは送信するデータの容量を低スペック計算機にはデータを少なくし高スペック計算機にはデータを多く送信することで処理速度の向上が出来るのではないかと考えられる。今後の課題としては、計算機のスペックに応じて画像を分割することにより、効率的な並列化を実現することが挙げられる。

参考文献

[1] 村田英明, “PVM3.4&リファレンスマニュアル”, 1995.
[2] 渡邊真也, “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
[3] The Message Passing Interface (MPI) Standard, http://www-unix.mcs.anl.gov/mpi/
[4] 早稲田大学理工学部コンピュータネットワーク工学科 山名研究室,
http://www.yama.info.waseda.ac.jp/~john/wiki
[5] 小野定康, 鈴木純司, “わかりやすい JPEG/MPEG2 の技術,” Ohmsha, 2001.
[6] 半谷精一郎, 杉山賢二, “JPEG・MPEG 完全理解,” コロナ社, 2005.

付録

以下に本研究で作成した並列 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;
}
```