卒業論文

PC クラスタ上での OpenMP による JPEG エンコーダの並列化

氏　　名：池上 広済
学籍番号：2210990211-3
指導教員：山崎 勝弘 教授

提 出 日：2003 年 2 月 21 日

立命館大学 理工学部 情報学科

内容梗概

並列処理は大規模な問題でも計算時間が大幅に短縮できるというメリットがあり，欠かせない技術の 1 つといえる．近年では，共有メモリ計算機の普及に伴い，並列プログラミングも分散メモリ環境から，共有メモリ環境へと移行しつつある．その共有メモリ用のプログラミングモデルとして現在注目を集めているのが OpenMP である．OpenMP は移植性が高く，プログラミングも比較的簡単なので，今後並列プログラミングの主流になると期待されている．また，高性能な PC が安価で手に入るようになり，Mrynet などの高速なネットワーク環境が普及してきた事から高性能な PC クラスタの構築が可能になった．

本論文では，並列処理の応用研究として本研究室で構築された SCore 型 PC クラスタ上での JPEG エンコーダの並列化について述べる．

JPEG エンコーダの並列化は，OpenMP を用いて行い，スレッド数 16 台で実行した．1024×768 画素と 1600×1200 画素の画像ファイルを圧縮した結果，1024×768 画素の画像を入力としてスレッド数１６台で実行した際，最高の 13．7 倍の速度向上が得られた．1600×1200 画素の画像ファイルは,1024×768 画素の画像ファイルよりも速度向上が低かった．これは,実行時にスワップが発生したためであると考えられる．

目次

図目次

表目次

１．はじめに

並列処理の研究の応用分野として気象予測，環境問題，流体計算，ディジタル画像処理，遺伝子の解明，データベース処理などが挙げられる．特に今後，データベース処理などとともに画像処理などのマルチメディアの分野で並列処理の活用が広がると考えられている．画像処理は，フィルタ処理のように同じ計算を大量の画素，フレームに適用したり，3DCG のように非常に計算量が多い処理が多く，並列処理に向いた性質がある．

並列計算機には 3 つのメモリモデルがある．複数のプロセッサがメモリバス/スイッチ経由で，主記憶に接続された共有メモリモデル(SMP)．プロセッサと主記憶から構成されるシステムが複数個互いに接続された形態で，プロセッサはほかのプロセッサの主記憶の読み書きができる，共有分散メモリ．プロセッサと主記憶からなるシステムが複数個互いに接続された形態で，プロセッサはほかのプロセッサの主記憶の読み書きができない，分散メモリがある．分散メモリに即した並列プログラミングライブラリとしては，PVM(Parallel Virtul Machine)，および，MPI(Message Passing Interface)が有名である．また，共有メモリに即した並列プログラミングモデルとしては，OpenMP が主流になってきている．

本研究では，並列処理のディジタル画像処理への応用の 1 つとして JPEG エンコーダを取り上げ，これを PC クラスタ上で並列実行した．JPEG のアルゴリズムは，入力画像に対して主に DCT 変換，量子化，エントロピー符号化を順に処理していく事から成り，それぞれの処理ごとに OpenMP を用いて並列化した．並列化アルゴリズムにはプロセッサファームを用いているが，JPEG エンコーダは 8×8 画素のブロック単位で処理を行うため，ブロック単位で分割し，並列実行した．

入力画像には 480×360 画素，1024×768 画素，1600×1200 画素の自然画像を用い，PC クラスタ上で 1 台から 16 台までのプロセッサを用いて実行時間を測定し，比較評価した．その結果，1024×768 画素の入力画像を用いて，プロセッサ 16 台で最高の 13．7 倍の速度向上が得られた．

2．PC クラスタ上での OpenMP プログラミング

2．1 PC クラスタ

PC クラスタとは，近年目覚しい速度で処理能力が高まり，大幅に低価格が進んだことにより，非常にコストパフォーマンスに優れた PC をネットワークで複数台接続し，全体としてのパフォーマンスを向上させるシステムである．1990 年前後に，数千から数万台の CPU を搭載する超並列計算機の開発が進む一方で，TCP/IP ベースのネットワークで接続された複数台の計算機を仮想的に 1 台のマシンとして捉えて並列プログラミングを走らせるような PVM(Parallel Virtual Machine)が開発された．PVM はそれぞれの計算機でメッセージ交換を行うメッセージ通信ライブラリであり，公開されたソフトウェアをインストールするだけで仮想的な並列処理環境が構築できた．これが PC クラスタの幕開けといえる．1995 年前後になると，イーサネットスイッチや Giga bit Ether などの技術も普及し，比較的安価に高性能なネットワークの構築が可能となった．さらに，Myricom 社の Myrinet などのクラスタを指向した専用の高速ネットワークが登場した．これにより，専用の並列計算機並みのスループットを持つネットワークの構築が可能となった．現在では複数のプロセッサを搭載した SMP 型の WS や PC が容易に入手できるようになり，これらをベースにした SMP クラスタが主流になっている．

本研究室の PC クラスタは，Pentium3 500MHz を搭載した PC16 台を 100Mbit/sec の通信速度を持つ Ethernet と 1,280Mbit/sec の通信速度を持つ Myrinet との 2 重のネットワークで接続されており，SCore と呼ばれるクラスタシステムソフトウェアを用いて通信を行っている．本研究室の PC クラスタの構成図を図 2 に示す[11].

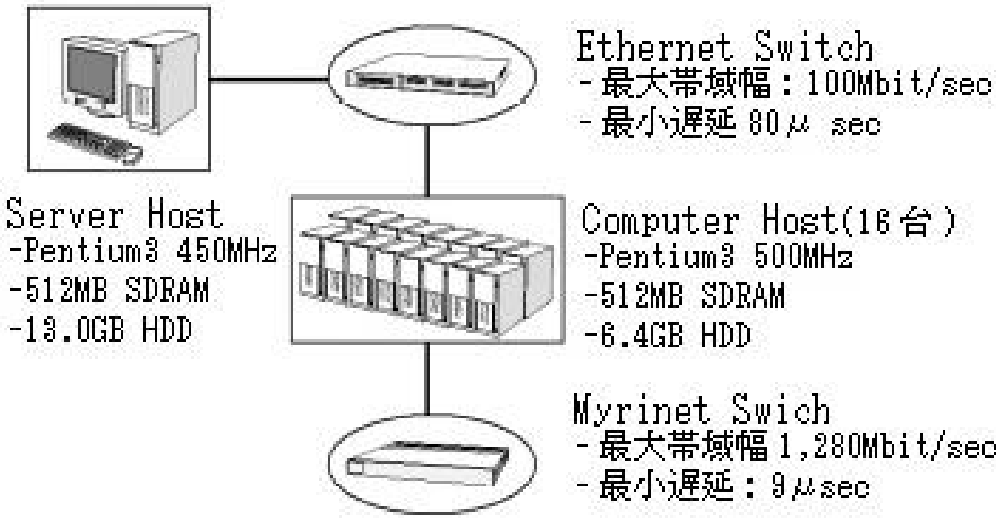

図 2：PC クラスタの構成図

SCore 型 PC クラスタは，RWCP(Real World Compurting Project)が開発した，高性能通信ライブラリを持つ SCore と呼ばれるクラスタシステムソフトウェアを用いて構築された、専用並列マシンと同等の性能を有するクラスタシステムである．SCore クラスタシステムソフトウェアの階層図を図 1 に示す．

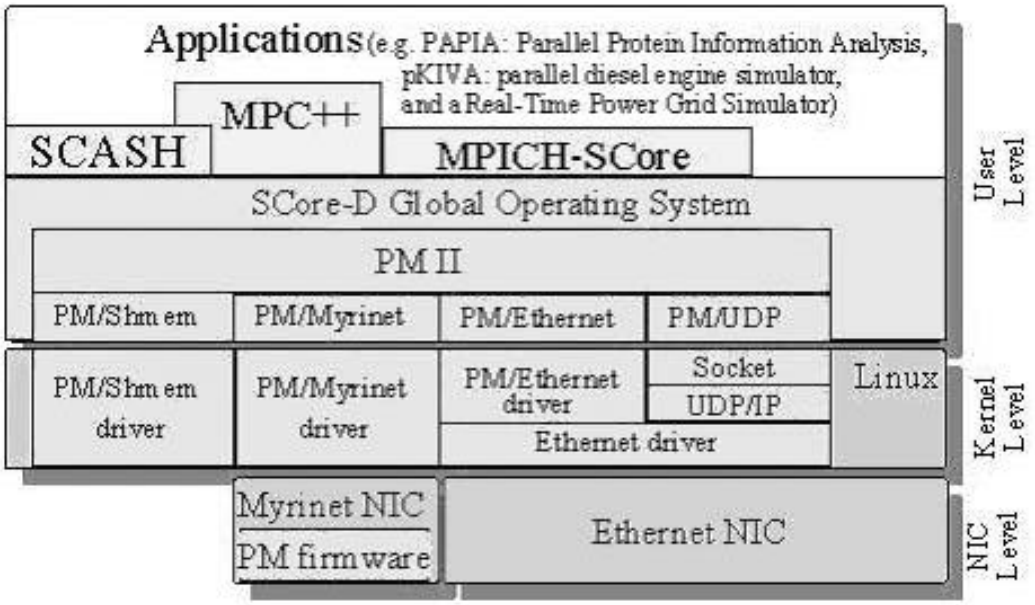

図 1：SCore クラスタシステムソフトウェアアーキテクチャ

以下に図 1 のソフトウェアについて簡単に説明する．

PMv2 通信ライブラリはクラスタコンピューティングのための低レベル通信ライブラリである．Ethernet,Myrinet,および共有メモリシステム上で動き，ユーザプログラミングで本通信ライブラリを利用する必要はない．MPI,PVM などの通信ライブラリが PMv2 上に実装されている．

SCore-D は，プロセッサやネットワークのような，クラスタのリソースを管理するグローバルオペレーティングシステムである．Linux カーネルを修正することなく，デーモンプロセスによって実現されていて，次の機能を提供している．

―ギャングスケジューリングによるマルチユーザ時分割空間分割スケジューリング

―実行時間ロードモニタ

―チェックポイント・リスタート機能

―対話型デバッガ起動

SCASH は，PMv2 を用いユーザレベルで実現したソフトウェア分散共有メモリシステムである．ソフトウェア分散共有メモリシステムは、共有メモリ型並列コンピュータのように、書くプロセッサからアクセスできる共有メモリ領域をソフトウェアで実現したシステムのことである[5].

２．２ OpenMP

OpenMP とは，共有メモリマルチプロセッサの並列プログラミングのためのプログラミングモデルであり，ベース言語(Fortran，C，C++)をコンパイラ指示文(directives，pragma)，ライブラリ，環境変数によって拡張したものである．並列実行や同期をプログラマが明示する事により並列化を行う．また，指示文を無視する事で逐次実行が可能なので逐次版と並列版を同じソースで管理でき，段階的な並列化が可能である．なお，本研究室の PC クラスタに実装されている OpenMP は，RWCP が開発した Omni OpenMP と呼ばれるもので，Fortran と C での並列化が可能である[3].

OpenMP は fork-join による並列実行モデルを用いている．OpenMP で記述されたプログラムは，マスタスレッドと呼ばれる単一のスレッドで実行を開始する．マスタスレッドは並列構文が現れるまで逐次リージョンを実行する．そして，並列指示文に遭遇すると複数のスレッドからなるチームを作成し，そのチームのマスタになる．ワークシェアリング構文に対応するブロックは，全スレッドによって実行されなければならない．全てのスレッドにより並列に実行される部分を並列リージョンという．また，ワークシェアリング構文に no wait 指示節が指定されていなければ，ワークシェアリング構文の最後で暗黙のバリア同期をチーム内の全スレッドに対して実行し，その後の処理はマスタスレッドのみが実行を続ける．1 つのプログラムにおいて，いくつでも並列構文を使う事ができる．したがって，プログラムは実行中の fork と join を繰り返す事になる．図 3 に fork-join モデルの実行の流れを示す[12].

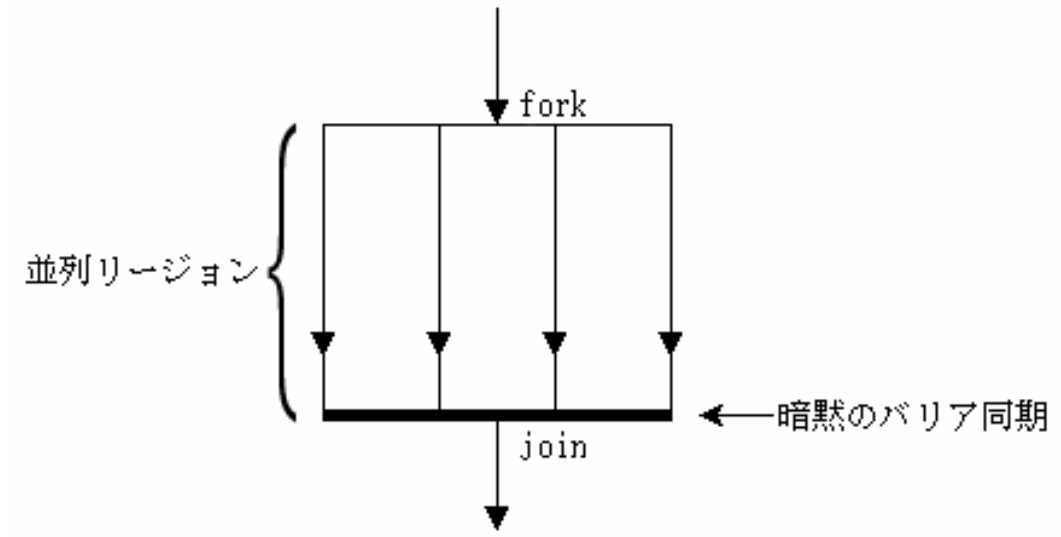

図 3：fork-join モデル

3． JPEG

3．1 JPEG とは

JPEG とは,“ Joint Photographic Experts Group ”の頭文字であり，カラー静止画像符号化標準方式を目指して，ISO(International Organization for Standardization)と CCITT(Consultative Committee for International Telegraph and Telephone)の 2 つの組織がジョイントして設立された検討グループを指している. 標準方式の正式名称は“ Digital compression and coding of continuous-tone still images ”であり，この名称が示す通りモノクロまたはカラーの連続的な諧調を有する静止画の符号化技術である.

JPEG は原画像を 1/10～1/100 程度に圧縮するが, 圧縮処理(エンコード)には通常, DCT, 量子化, エントロピー符号化という3 つの処理を順に行って実現する. そして, 圧縮された画像をコンピュータ上で見えるようにするための展開処理(デコード)には, 逆 DCT, 逆量子化, エントロピー復号化を順に行って実現する. 本研究では, 圧縮処理をするプログラムを作成し, OpenMP を用いて並列化した.

図 4 に JPEG のエンコードとデコードの流れを示す.

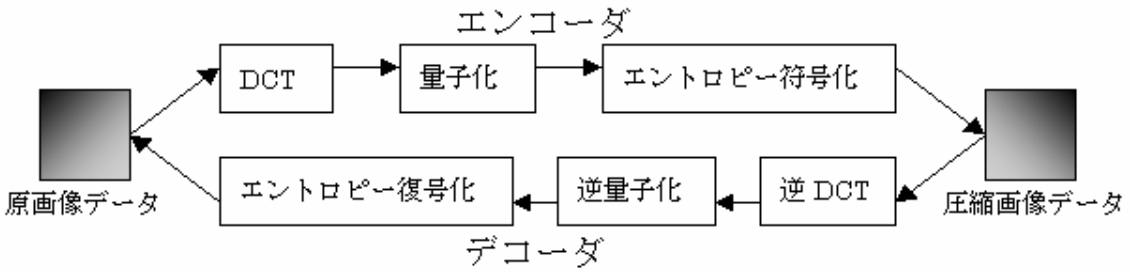

図 4：DCT ベースの JPEG のエンコードとデコード

3．2 動作モードと基本構成

JPEG は広い適用範囲を想定した汎用性の高いものであり, そのためにアルゴリズムの機能を 4 つの動作モードに分類して，ユーザがその中から必要な機能のみを選択できるようになっている．すなわち，ユーザは用途と必要な画質に応じて必要な機能や圧縮率を選択する事になる.

(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に 1 回のスキャンで行う．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

処理の順番および符号化処理は(1)の場合と同様であるが，処理スキャンの回数が複数回存在する．スキャンの順番が最初に近いほど，ブロック内の重要な情報であるといえる(す

なわち，大まかな画像が最初のスキャンで得られる).

(3) ロスレス(無ひずみ)

DCT 変換を用いず，近接画素との差分をエントロピー符号化するため，符号化によるひずみが生じない.

(4) ハイアラーキカル(階層化)

上記 3 つの動作モードを組み合わせて，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.

さらに，これらを具体的に実現する符号化アルゴリズムの基本構成によって分類すると，次に示すように DCT 方式と Spatial 方式の 2 つがある.

① DCT 方式

DCT 方式は，2 次元 DCT を行った係数を量子化したあとにエントロピー符号化する．このエントロピー符号化部として，ベースラインシステムはハフマン符号化，拡張システムはハフマン符号化のほかに算術符号化を用いることができる．算術符号化はハフマン符号化に比べて圧縮効率が若干高いが，それに伴って処理が複雑になる．DCT の演算は乗算を伴うため､たとえ量子化を行わなくても DCT 変換と逆変換で生じる演算の丸め誤差の相違，演算方法の相違などによって復号画像にひずみが加わる．しかし，発生するひずみを視覚的に目立たなくするための工夫がなされているため，圧縮率を大きくしても比較的高品質な画像を得ることができる.

② Spatial 方式

Spatial 方式は，2 次元空間上で隣接する画素値(最大 3 個)との差分を計算する DCPM 部とエントロピー符号化部とから構成される．エントロピー符号化には，ハフマン符号化と算術符号化の両者が選択できる．DPCM は DCT と異なり，乗算を含まず丸め誤差が発生しない．しがたって，Spatial 方式は，復号化後の画像が入力画像に完全に一致する無ひずみ圧縮である．インディペンデント機能(独立機能)と呼ばれ，符号化に DPCM(Differential PCM)とエントロピー符号化を使用しており，DCT 方式とのアルゴリズムの連続性はない．Spatial 方式は，符号化・復号化によってひずみが生じない可逆プロセスである.

表 1 に JPEG 符号化アルゴリズムの分類をまとめる.

表 1：JPEG 符号化アルゴリズムの分類

	DCT 方式		Spatial 方式
	ベースラインシステム	拡張システム	
符号化基本構成	DCT＋量子化＋エントロピー符号化		DPCM+エントロピー符号化
入力画像精度	8 ビット	8 ビットまたは 12 ビット	2～16 ビット
動作モード	シーケンシャル	シーケンシャルまたはプログレッシブ	シーケンシャル
エントロピー符号化	ハフマン符号化	ハフマン符号化または算術符号化	ハフマン符号化または算術符号化

なお, 本研究では最も基本的な JPEG 符号化アルゴリズムといえる DCT 方式のベースラインシステムを用いたエンコーダを作成した．次節以降は作成したエンコーダで採用したアルゴリズムの詳細である．

３．３ 離散コサイン変換(DCT)

画像の劣化には，目立ちやすいものと目立ちにくいものがあるが，目立ちにくいものとして，画像の高周波成分が欠けていることによる劣化がある．つまり画像の高い周波数成分は，多少削ってもそれにより画像品質の劣化が大きく目立つことは少ないということである．したがって，画像の高い周波数成分を多少削ることは，画像品質の劣化を抑えながら画像信号の帯域を狭める有効な方法となり，これは同時に画像データの容量を小さくする有効な方法となる．画像の高い周波数成分は画像の細かい絵柄の部分と，明暗の違いの大きい部分から発生する．したがって, 高い周波数成分を削るとこの部分で劣化を起こす．ただし，これは視覚的に目立たない劣化である．

この高い周波数成分を削るという処理の一つに DCT がある．JPEG の DCT 方式では, 最初に画像を 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．色差成分の圧縮において画素間引きを行って符号化する場合には，16×16 画素または 8×16 画素の領域から間引きによって 8×8 画素のブロックを作成する．

ブロック内の 2 次元データを $S_{yx}(y, x = 0,1,2,\cdots, y)$ ， DCT 変換係数を $S_{vu}(y, x = 0,1,2,\cdots, y)$ とすると，DCT 変換および逆変換は，それぞれ以下の式で定義される．

DCT 変換　：$S_{vu} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} \cos\frac{(2x+1)up}{16} \cos\frac{(2y+1)up}{16}$

DCT 逆変換：$S_{yx} = \frac{1}{4} \sum_{x=0}^{7} \sum_{y=0}^{7} C_u C_v S_{vu} \cos\frac{(2x+1)up}{16} \cos\frac{(2y+1)vp}{16}$

ここで

$$C_u, C_v = \begin{cases} 1/\sqrt{2} & for \quad u, v = 0 \\ 1 & otherwise \end{cases}$$

DCT 変換数 S_{vu} は, 1 つのブロックに対して 64 個存在するが, このうち S_{00} を DC 成分、残りの 63 個を AC 成分と称し，v または u が大きいほど高い空間周波数成分に相当する．

３．４ 量子化

DCT 変換のあとに，64 個の DCT 係数S_{vu}は量子化という操作によってとりうるレベル数が制限される．各 DCT 係数の量子化は，定められた量子化ステップサイズの何倍であるかを四捨五入した数値で求めることである．すなわち，量子化された 2 次元 DCT 係数の量子化ステップサイズを$Q_{vu}(v,u=0,2,3,\cdots,7)$，量子化された 2 次元 DCT 係数を$Sq_{vu}(v,u=0,1,2,\cdots,7)$とすれば，量子化および逆量子化は，それぞれ以下の式で定義される．

量子化　：$Sq_{vu}=round\left(\frac{S_{vu}}{Q_{vu}}\right)$

逆量子化：$R_{vu}=Sq_{vu}\cdot Q_{vu}$

ここで，round は最も近い数値への四捨五入を指す．

量子化ステップサイズの最適値は，入力画像および画像の表示装置の特性に依存して定まるため，JPEG では具体的な値を規定していない．しかし，JPEG 規定の Annex K のガイドラインに輝度成分と色差成分に対する量子化テーブルの例が示されている．

表 2：量子化ケーブルの例

輝度成分用量子化テーブル

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

色差成分用量子化テーブル

17	18	24	47	99	99	99	99
18	21	56	66	99	99	99	99
24	26	99	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

３．５ エントロピー符号化

量子化のあとには，エントロピー符号化が行われる．ここではエントロピー符号化の 1 手法であるハフマン符号化について説明する．

DCT 方式のハフマン符号化は，64 個の DCT 係数に対し DC 成分と AC 成分で別々に行われる．DC 成分の符号化は，隣接する直前のブロックの DC 成分との差分がハフマン符号化される．AC 成分に対しては，量子化によって大部分の係数値が 0 となることから，非零の AC 成分とその AC 成分の前に位置する零値の AC 成分の個数を組み合わせ，零ランレングスを考慮したハフマン符号化が行われる．以下，具体的な符号化手順について説明する．

(1) ジグザグスキャンによる並べ替え

2 次元の DCT 変換係数をハフマン符号化するために，最初に 2 次元配列から 1 次元配列への変換，すなわちジグザグスキャンが行われる．8×8 の DCT 変換係数からなる 2 次元信号を $Sq_{vu}(v,u=0,1,2,\cdots,7)$ とし，これをさらに表 3 に示した順序でジグザグスキャンして 1 次元信号に変換した信号を $ZZ(k)(k=0,1,2,\cdots,63)$ とし，ZZ(0)が DC 成分，ZZ(1)～ZZ(63)が AC 成分である．DC 成分は、直前における DC 成分の量子化値との差分信号 Diff をハフマン符号化する．また AC 成分は，ZZ(1)，ZZ(2)，･･･，ZZ(63)の連続する零の個数と非零の AC 成分を組み合わせてハフマン符号化する．

表 3：DCT 係数のジグザグスキャンの順番

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

(2) DC 差分値の符号化

DC 差分値 Diff の符号化を行うには，最初に表 4 に示される Diff の値に対応するカテゴリ SSSS とハフマン符号を求める．(表 4 は JPEG 規定の Annex K による輝度成分に対するハフマン符号の例).

表 4：DC 差分値符号化用ハフマン符号の例と付加ビット

Diff	SSSS	ハフマン符号	付加ビット
-2047…-1024,1024…2047	11	111111110	00000000000…01111111111,10000000000…11111111111
-1023…-512,512…1024	10	11111110	0000000000…0111111111,1000000000…1111111111
-511…-256,256…511	9	1111110	000000000…011111111,100000000…111111111
-255…-128,128…255	8	111110	00000000…01111111,10000000…11111111
-127…-64,64…127	7	11110	0000000…0111111,1000000…1111111
-63…-32,32…63	6	1110	000000…011111,100000…111111
-31…-16,16…31	5	110	00000…01111,10000…11111
-15…-8,8…15	4	101	0000…0111,1000…1111
-7…-4,4…7	3	100	000…011,100,111
-3,-2,2,3	2	11	00,01,10,11
-1,1	1	10	0,1
0	0	00	なし

DC 差分値のハフマン符号化は，DC 差分値の大きさのカテゴリを最初にハフマン符号で指定し，その後に続く付加ビットで，カテゴリ内の DC 差分値の具体的な値を指定する形，すなわち(カテゴリのハフマン符号)＋(付加ビット)の順番に結合された可変長の符号を用いて行われる．

(3) AC 成分の符号化

AC 成分の符号化は，非零の AC 成分とその非零係数の前にある連続する零の AC 成分の個数を組み合わせて可変長符号を構成し，さらに DC 差分値の符号化と同様に付加ビットを加える．具体的な符号化手順は，最初に AC 成分を ZZ(1)から順番に見ていき，非零の AC 成分があれば，それ以前の零値の AC 成分の個数を 4 ビットの RRRR とし，さらに現在の非零の AC 成分に対するカテゴリを 4 ビットの SSSS(SSSS の求め方は DC 差分値符号化の場合と同様)として表し，これらの組み合わせを用いて表 5 の中で対応するハフマン符号を求める(表 5 は JPEG 規定の Annex K による輝度成分に対するハフマンの例)．このように，零値のランレングスを加味した可変長符号化を行うことにより，連続する零値の個数を一括して 1 つの符号で表現できるため，高い符号化効率が得られる．

表 5：AC 成分符号化用ハフマン符号の例

RRRR/SSSS	0	1	2	…	10
0	1010(EOB)	00	01	…	1111111110000011
1	なし	1100	11011	…	1111111110001000
2		11100	11111001	…	1111111110001110
…		…	,,,	…	…
15	11111111001（ZRL）	111111111110101	1111111111110110	…	1111111111111110

以上述べた AC 成分の処理には 2 つの特別な場合がある．そのうちの 1 つは零値の AC 成分が 16 個以上連続する場合である．すなわち，零ランレングスを表現するための RRRR が 4 ビットであることにより 16 以上の例ランレングスは表現できない．そのため，連続する零値が 15 以下となるまで 16 個の零のグループに対して 1 つの RRRR/SSSS=15/0 の組み合わせに対する符号 ZRL(Zero Run Length)を出力する．もう 1 つは，最後の AC 成分 ZZ(63)が 0 の場合であり，このときはブロックの最後に残った零値の AC 成分に対して一括して RRRR/SSSS=0/0 の組み合わせに対応する符号 EOB(End Of Block)を出力した後，直ちに当該ブロックの符号化処理を終了する．そして，ZRL または EOB 以外のハフマン符号の後には，具体的な AC 成分の大きさを特定するための付加ビットを加える．[6]

復号化の手順は，符号化処理を逆にたどればよい．

(4) DC 成分の復号化

DC 成分の復号化は，次の 3 ステップからなる．

①ビットストリームから対応するハフマン符号を見つけて復号化し，符号化要素のカテゴリ SSSS を得る．

②引き続いてビットストリームから SSSS ビット分の符号化ビットを読み込み，DC 差分値 Diff とする．

③Diff の先頭ビットが 0 の場合は DC 差分値が負であるため，Diff+1 を求めた後に SSSS+1 ビット目の LSB 以上に“ 1 ”をつめて負数に変換する．

最終的な DC 成分 ZZ(0)の再生には，上記手順によって得られた DC 差分値 Diff に，直前のブロックにおける DC 成分の再生値 Pred を加えることによって得られる．

すなわち，

$$ZZ(0) = Diff + \mathrm{Pred}$$

である．

(5) AC 成分の符号化

AC 成分の符号化は DC 成分の符号化よりやや複雑で，以下の 6 ステップからなる．

①AC 成分の配列 $ZZ(i)(i \le 1)$ をすべて 0 にリセットする．

②ビットストリームから対応するハフマン符号を見つけて復号化し，零ランレングスと非

零 AC 成分のカテゴリの組み合わせとなっている符号化要素シンボル RS を得る.

④ZRL が検出されたら AC 成分のアドレスを 16 個分更新して処理②に戻る．これは，16 個分の AC 成分を 0 とすることと等しい.

⑤EOB が検出されたら当該ブロックの復号化処理を終了する．これは，残りの AC 成分をすべて 0 とすることに等しい.

⑥SSSS が 0 でなければ付加ビットを読み出し，付加ビットの先頭ビットが 0 の場合は AC 成分が負であるため，AC+1 を求めた後に SSSS+1 ビット目の LSB 以上に“①”をつめて負数に変換する．この時点で，DCT 係数のアドレスが 63 に達していれば複合化処理を終了し、63 に満たなければ処理②に戻る.

(6)ハフマン符号の特定

DC 成分の復号化手順①と AC 成分の復号化手順②で，ビットストリームから対応するハフマン符号を探し出す処理での中で最も単純な方法は，ビットストリームから 1 ビットずつ読み出して，符号器用ハフマンテーブルを参照しながら次々にビットパターンマッチングを行う方法であるが，この方法は処理量が膨大である．したがって，ハフマン符号をその発生頻度順に並べ替えた復号器用テーブルを準備することによって，効率的な符号検索を行うことができる.

復号器用テーブルとして 4 種類のテーブル(MINCODE，MAXCODE，VALPTR，HUFFVAL）を符号器用ハフマンテーブルに基づいて作成する．最初に，符号器で使用したハフマン符号を 2 進数で表現された整数値であるとみなし，その大きさの小さい順番に並べる．このテーブルに基づきビット長が I のハフマン符号に対して，MINCODE(I)，MAXCODE(I)，VALPTR(I)のテーブルを作成する．ここで，MINCODE(I)，MAXCODE(I)は I ビットのハフマン符号の最小値と最大値，VALPTR(I)は，MINCODE(I)が大きさ順のテーブルの中で何番目かを示すポインタである．さらにハフマン符号が J 番目に小さい符号値を並べたテーブル HUFFVAL(J)を作成する．表 6 と表 7 に DC 成分と AC 成分に対して得られる VALPTR(J)，MINCODE(I)，MAXCODE(I)の例を示す.

そして，ビットストリームから対応するハフマン符号を探し出して復号化するためのアルゴリズムは，以下の 3 ステップから構成される.

①ビットストリーム中のハフマン符号から 1 ビット分の符号 CODE を読み込み，I=1 とする.

②CODE と MAXCODE(I)を大小比較し，CODE<MAXCODE(I)の条件が成立すれば，これは復号化すべきハフマン符号の長さが I ビットであることを示している．このとき復号値のテーブル HUFFVAL(J)のアドレス J は，HUFFVAL テーブルの中でハフマン符号が I ビットである符号値が置かれる開始ポインタ VALPTR(I)に，実際のハフマン符号が I ビット符号の中で何番目の大きさかを表す CODE－MINCODE(I)を加えることにより，J=VALPTR(I)+CODE－MINCODE(I)として求まる．したがって，復号値は HUFFVAL(J)

であり，復号化が終了する．

③CODE>MAXCODE(I)の場合には，符号化すべきハフマン符号のビット長が I ビット以上であるため，ビットストリームから引き続いて 1 ビットを読み出して現在の CODE の LSB の後に付加し，I+1 ビット分の CODE を作成する．さらに I+1 を新しい I として手順②へ戻る[6].

表 6：DC 成分の復号化テーブル

I	VALPTR(I)	MINCODE(I)	MAXCODE(I)
1	-	-	1111111111111111
2	0	00	00
3	1	010	110
4	6	1110	1110
5	7	1110	11110
6	8	11110	111110
7	0	111110	1111110
8	10	1111110	11111110
9	11	11111110	111111110

表 7：AC 成分の復号化テーブル

I	VALPTR(I)	MINCODE(I)	MAXCODE(I)
1	-	-	1111111111111111
2	0	0	01
3	2	100	100
4	3	1010	1100
5	6	11010	11100
6	9	111010	111011
7	11	1111000	1111011
8	15	11111000	11111010
9	18	111110110	111111010
10	23	1111110110	1111111010
11	28	11111110110	11111111001
12	32	111111110100	111111110111
13	-	-	1111111111111111
14	-	-	1111111111111111
15	36	111111111000000	111111111000000
16	37	1111111110000010	1111111111111110

４．OpenMP による JPEG エンコーダの並列処理

４．１ JPEG エンコーダの並列化アルゴリズム

４．１．１ JPEG エンコーダの並列化アルゴリズムの手順

OpenMP を用いているため，並列化アルゴリズムは DCT 変換，量子化，ハフマン符号化の各ステップで fork と join を繰り返している．DCT 変換と量子化のステップは，処理が各 8×8 画素のブロックごとに独立して行えるため，ブロック単位に画像を分割すれば並列化して処理を行える．しかし，ハフマン符号化のステップでは，処理は完全に独立しておらず，AC 成分の処理は DC 成分の処理に依存し，DC 成分は隣接する DC 成分に依存している．そのため，DC 成分は並列化ができず，AC 成分はブロック単位で分割する必要がある．

実行時間は，原画像の入力後からハフマン符号化が完了するまでを測定した．図 5 に JPEG エンコーダの並列化アルゴリズム全体の流れを示す．

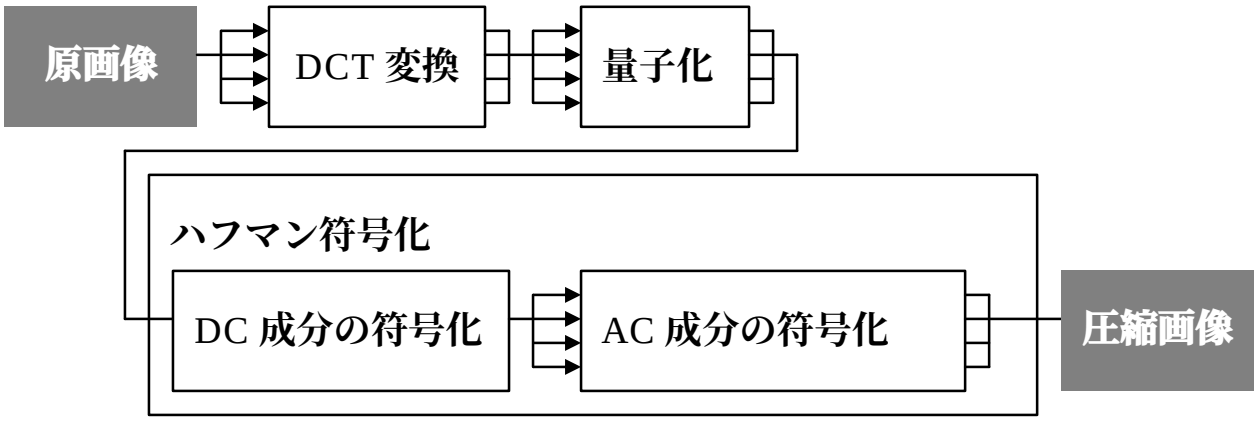

図 5：並列化アルゴリズムの全体の流れ

４．１．２ DCT・量子化の並列化アルゴリズム

DCT 変換および量子化では，各ブロックの画素ごとに独立に３．３節および３．４節で与えられた式を解くことにより処理している．そのためブロックで区切られるべき場所で区切らなければ，並列化した状態で処理することができない．しかし，今回サンプルに 2 の乗数の解像度を持つ画像を使用したため，2 の乗数のプロセッサ台数で実行する場合には特別な処理は必要ない．そのため，並列化アルゴリズムはプロセッサファームを用いた．CPU4 台で実行する場合，図 6 のように画像をブロック分割で 4 つに区切り、それぞれを CPU に割り当て並列化している．

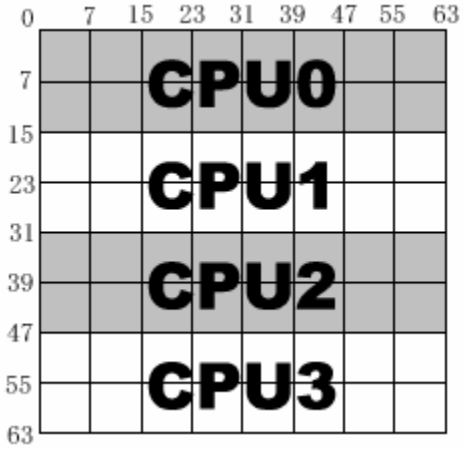

図 6：ブロック単位での画像分割方法

４．１．３ ハフマン符号化の並列化アルゴリズム

ハフマン符号化では，DC 成分の符号化は隣接する直前のブロックの DC 成分との差分値をとって順次符号化していくため，最初の画素の値が最後の画素の値に依存される．そのため画像を分割してしまうと，違うプロセッサで処理している部分の画像の DC 成分を参照できない．ゆえに，並列化する前に DC 成分の処理をしておき，それを参照して AC 成分の処理を並列に処理させた．AC 成分の符号化は，ブロック単位で独立して処理を行えるため，DCT 変換，量子化と同様の方法で並列化している．

４．２ 実行結果

図 8 のような原画像を入力とし，PC クラスタ上で実行した．原画像は bitmap 形式の画像で，480×360 画素，1024×768 画素及び 1600×1200 画素の 3 種類を実行し，圧縮率はそれぞれ約 6．8%になった．

図 7：原画像

JPEG エンコードの実行結果を以下に示す.

表 8 :解像度 480×360 での実行時間

スレッド数	1	2	4	8	16
DCT	8.347	4.340	2.170	1.184	0.610
量子化	0.098	0.061	0.032	0.036	0.030
ハフマン符号化	0.186	0.129	0.068	0.046	0.032
合計	8.720	4.611	2.341	1.334	0.737

単位 :秒

表 9 :解像度 1024×768 での実行時間

スレッド数	1	2	4	8	16
DCT	38.625	19.850	9.927	4.963	2.486
量子化	0.434	0.264	0.133	0.069	0.037
ハフマン符号化	0.824	0.571	0.295	0.183	0.114
合計	40.254	21.035	10.668	5.509	2.917

単位 :秒

表 10 :解像度 1600×1200 での実行時間

	1	2	4	8	16
DCT	93.034	47.874	23.941	11.974	6.410
量子化	1.086	0.671	0.335	0.170	0.226
ハフマン符号化	2.099	1.428	0.734	0.455	0.311
合計	97.257	50.884	25.797	13.320	7.633

単位 :秒

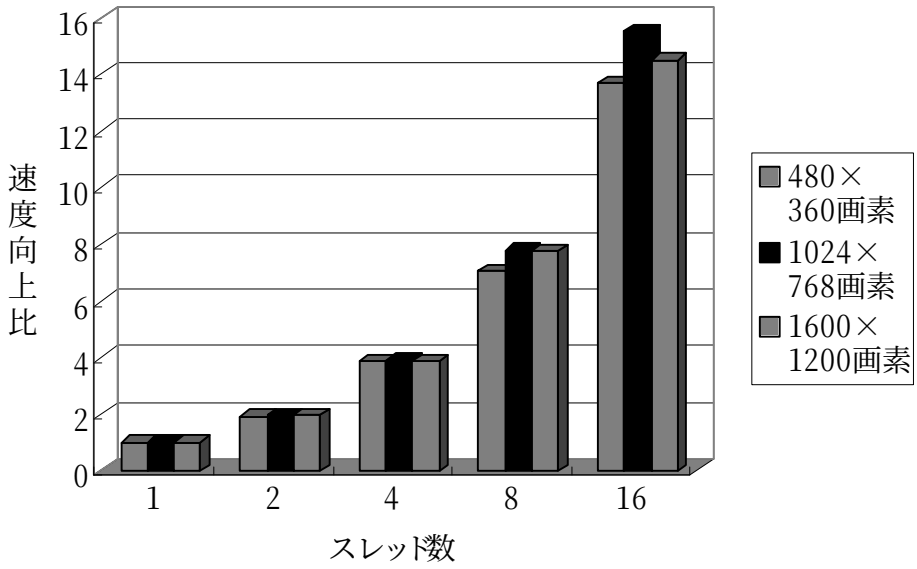

図 8：DCT の速度向上比

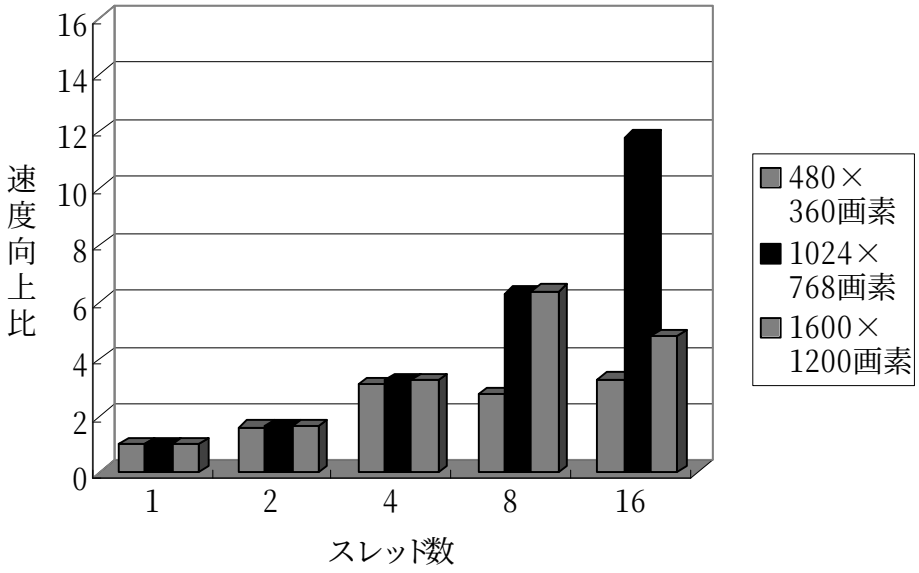

図 9：量子化の速度向上比

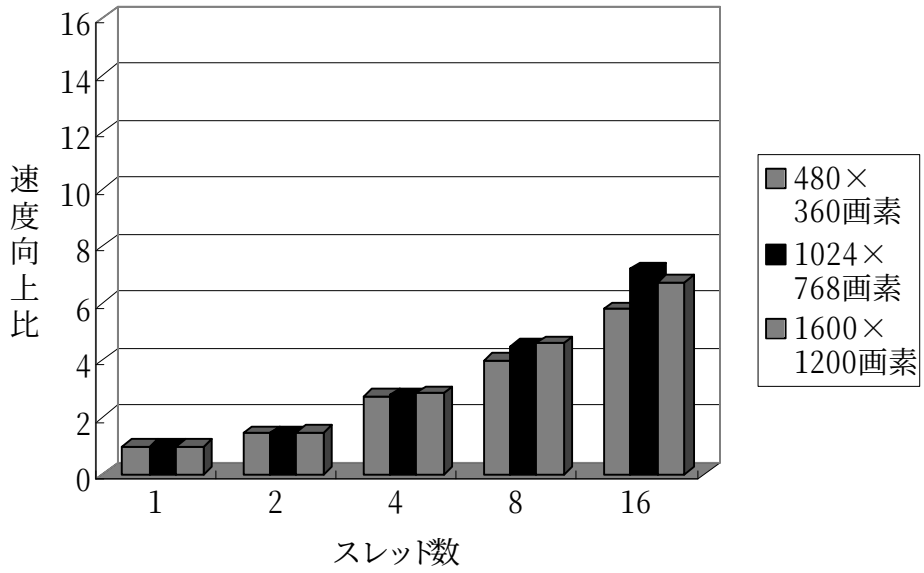

図 10：ハフマン符号化の速度向上比

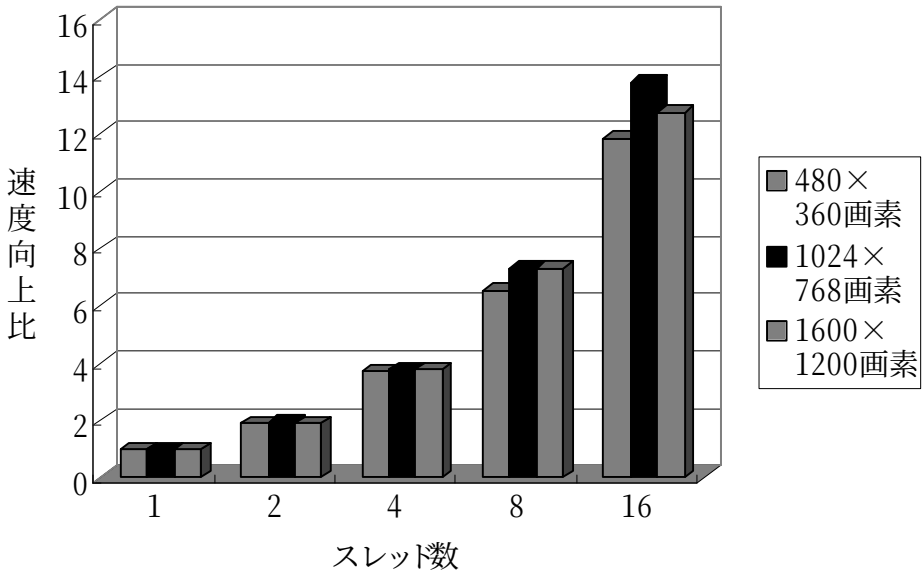

図 11：JPEG エンコーダ全体での速度向上比

４．３ 考察

実行結果は，スレッド数 16 台で 480×360 画素で 11．8 倍，1024×768 画素で 13.7 倍，1600×1200 画素で 12.7 倍となった．また，各ステップごとに比較してみると処理時間のほとんどが DCT 変換にかかっていた事がわかる．そして，DCT 変換の速度向上比は他のステップに対しても圧倒的に高い．このことより，負荷の高い処理では並列化効果が高まるということがわかる．これは，画像の分割やプロセッサ間の通信等にかかる時間が実際の計算時間に対して極めて小さいからである．量子化，ハフマン符号化のステップでは，プロセッサ 1 台でも実行時間が 1 秒もかからないため，並列化効果はあまり得られなかった．

ここで，ファイルサイズごとに速度向上比を比較してみると，1024×768 画素での実行結果が最も並列化効果が高く，480×360 画素での実行結果が最も並列化効果が低い事がわかる．通常，ファイルサイズが大きくなれば，実際の処理にかかる時間が，画像の分割やプロセッサ間の通信等にかかる時間に対して大きくなるので，並列化効果が高まる．1600×1200 画素での並列化効果があまり高くならなかった原因は，実行の際に物理メモリが不足しスワップが起こっていた事が原因と思われる．そして，スワップが起こらない範囲では，画像が大きい方がより高い速度向上が得られる事がこの実行結果からいえる．

また，1600×1200 画素での量子化でスレッド数 16 台での結果が 8 台での結果よりも下回る結果となった．他のステップではこういった現象は見られず，図 8 とは違う画像を用いた場合にも同様に実行速度が遅くなるため，量子化のアルゴリズムが影響しているものと考えられる．

５．おわりに

本研究では, 本研究室の SCore 型 PC クラスタ上で OpenMP によって JPEG エンコーダを並列化し，実行した．その結果，1024×768 画素の画像で，最高の約 13．7 倍の並列化効果を得られた．1600×1200 画素で実行した場合，物理メモリが不足してスワップが起こるため，1024×768 画素以上の並列化効果は得られず，それ以上の大きな画像では実行できなかった．これは scash の制約が原因で，本研究室の PC クラスタでは 180MB 程度しか確保できない事が確認されている．

今後の課題としては，JPEG デコーダを作成し，実行結果を確認しなければならない．そして，静止画像に比べてより大容量になる動画像への並列処理の応用が考えられる．しかし，コンピュータホストの物理メモリが 512MB の SCore 型 PC クラスタの場合，実際のプログラムの中では 180MB 程度しか確保できないため, より大容量となる動画像を扱う際には, このメモリの制約を乗り越えるプログラミングができればよいと思われる．そして，SCore のバージョンがアップされるとともに，OpenMP のコンパイラや，SCASH の性能も良くなっていけば，PC クラスタ上での OpenMP による並列プログラミングは今後ますます実用化されていくことと思われる．

謝辞

本研究の機会を与えてくださり，数々の助言をいただきました山崎勝弘教授ならびに小柳滋教授に心より感謝いたします．また本研究にあたり，励ましの言葉や貴重なご意見をいただきました三浦誉大様，宮城雅人様，また本研究室の皆様にも心より感謝いたします．

参考文献

[1] 湯浅　太一、安村　通晃、中田　登志之：“bit 別冊　はじめての並列プログラミング”、共立出版、1998.

[2] PC Cluster Consortium：http://www.pccluster.org/.

[3] 佐藤　三久：“OpenMP”、JSPP'99　OpenMP　チュートリアル資料、1999.

[4] Rohit Chandra, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, Ramesh Menon：“Parallel Programming in OpenMP”、Morgan Kaufmann Publishers、2001.

[5] 石川 裕，佐藤 三久，堀 敦史，住元真司，原田 浩，高橋俊行:Linux で並列処理をしよう，共立出版，2002

[6] 小野　定康、鈴木　順司：“わかりやすい JPEG/MPEG2 の技術”，オーム社、2001.

[7] Steve Oualline，(監訳)望月　康司：“Practical C Programming”、オライリー・ジャパン、1998.

[8] Peter van der Linden、(訳)梅原 系：“エキスパート　C プログラミング　知られざる C の深層”、ASCII 出版、1996.

[9] 内田　大介：“OpenMP による並列プログラミング１”、立命館大学理工学部情報学科卒業論文、2000.

[10] 土屋　悠輝：“OpenMP による並列プログラミング２”、立命館大学理工学部情報学科卒業論文、2000.

[11] 三浦　誉大：“PC クラスタ上での並列プログラミング環境の構築”、立命館大学理工学部情報学科卒業論文、2002.

[12] 大村　浩文：“PC クラスタの動作テストと OpenMP 並列プログラミング”、立命館大学理工学部情報学科卒業論文、2002.

[13] 宮城　雅人：“PC クラスタ上での OpenMP による JPEG2000 エンコーダの並列化”、立命館大学理工学部情報学科卒業論文、2003.

付録

```
void dct(void){
 int  u,v;
 double cu,cv;
 double sum_y, sum_u, sum_v;

#pragma omp parallel for
private(y,x,v,u,cv,cu,sum_y,sum_u,sum_v,cos_x,cos_y,y_block,x_block)
 for(y_block = 0; y_block < (Y_SIZE>>3); y_block++){
  for(x_block = 0; x_block < (X_SIZE>>3); x_block++){
        for(v = 0; v < BLOCK; v++){
                if( v == 0)
                        cv = 1/sqrt(2.0);
                else
                        cv = 1;

         for(u = 0; u < BLOCK; u++){
                if( u == 0)
                        cu = 1/sqrt(2.0);
                else
                        cu = 1;

                sum_y = 0;
                sum_u = 0;
                sum_v = 0;

                        for(y = 0; y < BLOCK; y++){
                         for(x = 0; x < BLOCK; x++){
                         cos_x = cos( ((2.0*x+1.0)*u*M_PI)/16 );
                                cos_y = cos( ((2.0*y+1.0)*v*M_PI)/16 );
                         sum_y += image_y[y_block*BLOCK+y]
                                                [x_block*BLOCK+x]*cos_x*cos_y;
                         sum_u += image_u[y_block*BLOCK+y]
                                                [x_block*BLOCK+x]*cos_x*cos_y;
```

```
                        sum_v += image_v[y_block*BLOCK+y]
                                        [x_block*BLOCK+x]*cos_x*cos_y;
                    }
                }

            image_y_dct[y_block*BLOCK+v]
                                        [x_block*BLOCK+u] = (cu*cv*sum_y)/4;
            image_u_dct[y_block*BLOCK+v]
                                        [x_block*BLOCK+u] = (cu*cv*sum_u)/4;
            image_v_dct[y_block*BLOCK+v]
                                        [x_block*BLOCK+u] = (cu*cv*sum_v)/4;
             }
        }
    }
 }
}

void quantization(void){
#pragma omp parallel for private(y_block,x_block,y,x)
 for(y_block = 0; y_block < (Y_SIZE>>3); y_block++){
  for(x_block = 0; x_block < (X_SIZE>>3); x_block++){
      for(y = 0; y < BLOCK; y++){
       for(x = 0; x < BLOCK; x++){

          image_y_dct[y_block*BLOCK+y][x_block*BLOCK+x] /= q_table_y[y][x];
          image_u_dct[y_block*BLOCK+y][x_block*BLOCK+x] /= q_table_uv[y][x];
          image_v_dct[y_block*BLOCK+y][x_block*BLOCK+x] /= q_table_uv[y][x];

       }
      }
  }
 }
}
```

```
void HuffmanEnc(void){

   int count;
   int Ac_x,Ac_y;

   PreDc_Y = 0;
   PreDc_U = 0;
   PreDc_V = 0;

   run_y = 0;
   run_u = 0;
   run_v = 0;

#pragma omp parallel for
  private(y_block,x_block,y,x,count)
shared(Dc_Y,PreDc_Y,Dc_U,PreDc_U,Dc_V,PreDc_V,run_y,run_u,run_v,bitcountor)
   for(y=0;y<Y_SIZE;y+=BLOCK){
      for(x=0;x<X_SIZE;x+=BLOCK){

   /* Y 成分の符号化 */
        /* DC 成分の符号化 */
          Dc_Y = image_y_dct[y][x] - PreDc_Y;
          PreDc_Y += Dc_Y;
          EncodeDc(Dc_Y);

        /* AC 成分の符号化 */

          for(count=1;count<64;count++){
        Zigzag(count,y,x,&Ac_y,&Ac_x);
            Ac = image_y_dct[Ac_y][Ac_x];
                    if(Ac == 0){
                       if(count >= 63)
                          EncodeAc(0,0);
                       run_y += 1;
                    }
                    else{
```

```
                while(run_y > 15){
                  EncodeAc(0,15);
                  run_y -= 16;
                }
                EncodeAc(Ac,run_y);
                if(count >= 64) break;
                run_y = 0;
              }
  }

/* U 成分の符号化 */

  Dc_U = image_y_dct[y][x] - PreDc_U;
  PreDc_U += Dc_U;
  EncodeDc(Dc_U);

  /* AC 成分の符号化 */

  for(count=1;count<64;count++){
 Zigzag(count,y,x,&Ac_y,&Ac_x);
    Ac = image_y_dct[Ac_y][Ac_x];
            if(Ac == 0){
              if(count >= 64)
                EncodeAc(0,0);
              run_u += 1;
            }
            else{
              while(run_u > 15){
                EncodeAc(0,15);
                run_u -= 16;
              }
              EncodeAc(Ac,run_u);
              if(count >= 64) break;
              run_u = 0;
            }
  }
```

```
        /* V 成分の符号化 */
          Dc_V = image_y_dct[y][x] - PreDc_V;
          PreDc_V += Dc_V;
          EncodeDc(Dc_V);

          /* AC 成分の符号化 */

          for(count=1;count<64;count++){
         Zigzag(count,y,x,&Ac_y,&Ac_x);
          Ac = image_y_dct[Ac_y][Ac_x];
                    if(Ac == 0){
                      if(count >= 64)
                        EncodeAc(0,0);
                      run_v += 1;
                    }
                    else{
                      while(run_v > 15){
                        EncodeAc(0,15);
                        run_v -= 16;
                      }
                      EncodeAc(Ac,run_v);
                      if(count >= 64) break;
                      run_v = 0;
                    }
          }
          }
        }
}
```