修士論文

ハイブリッド並列プログラミングによる MPEG2 エンコーダの高速化

氏　　名	:	池上　広済
学籍番号	:	6124040013-7
指導教員	:	山崎　勝弘　教授
提出日	:	2006 年 2 月 15 日

立命館大学大学院　理工学研究科　情報システム学専攻

内容梗概

本論文では，MPEG2 エンコーダを対象として，SMP クラスタ上でハイブリッド並列プログラミングと MPI による並列プログラミングの 2 通りで並列化をし，それらの性能を比較して，ハイブリッド並列プログラミングの有用性を示す.

本研究では，1 ノード内に複数のプロセッサが搭載された対称型マルチプロセッサ SMP クラスタを有効活用するために，メッセージ通信インタフェースライブラリの MPI と共有メモリプログラミングモデルの OpenMP を併用したハイブリッド並列プログラミングによる高速化について検証を行った.

実験に用いるプログラムは，MPEG Software Simulation Group によるリファレンス実装である MPEG-2 Encoder version 1.2 を用いた．ノード間は MPI により並列化し，各ノードで GOP（Group Of Pictures 通常 15 フレーム)単位の動画をエンコードする．ノード内では OpenMP により処理の重い離散コサイン変換（DCT)，逆 DCT，量子化，逆量子化を並列化する.

64 秒間の動画を対象に, バイブリッド並列プログラミングの方が MPI による並列プログラミングより 10～20%高速で，速度向上は最大で 16 ノード 13.9 倍の結果が得られた．また，サイクリック分割による並列化の方がブロック分割に比べ約 10%高速であるという結果が得られた.

本論文では，まず SMP クラスタとハイブリッド並列プログラミング，MPEG2 エンコーダのアルゴリズムについて述べ，並列化の考案，実装，評価を行う.

目次

図目次

表目次

1. はじめに

近年，ノード内に複数のプロセッサを搭載したSMP（対称型マルチプロセッサ）構造を持つSMPクラスタが価格・性能比に優れた並列計算機として期待されている．並列計算機には，複数のプロセッサがメモリを共有する共有メモリモデル（SMP)，プロセッサとメモリから構成された計算機が複数台接続され，プロセッサは他の計算機のメモリにアクセスできない分散メモリモデル，及びプロセッサが他の計算機のメモリにアクセスでき，共有メモリモデルのように動作する分散共有メモリモデルの3つがある．分散メモリモデルでの並列プログラミングには，メッセージ通信ライブラリであるMPI（Message Passing Interface）やPVM（Parallel virtual Machine）が広く使われており，共有メモリモデルでの並列プログラミングには，共有メモリ並列プログラミングモデルであるOpenMPが主流となってきている．

SMPクラスタは，共有メモリモデルと分散メモリモデルが混在したアーキテクチャである．現在，並列プログラミングの主流となっているMPIでは，SMPノード内部でもメッセージ通信を行うため，オーバーヘッドが大きい．また，分散共有メモリモデルによりOpenMPで並列化した場合，ノード間のメモリの同期はソフトウェアでの処理になるため，ノード間のデータ共有はMPIよりもオーバーヘッドが大きくなる．よって，SMPクラスタ上ではノード間はMPIで通信し，ノード内ではOpenMPによりデータ共有を行う，ハイブリッド並列プログラミングにより並列化を行うことが理想である．

一方，コンピュータやハードディスクの低価格化・高性能化により，放送番組制作においてビデオ編集をコンピュータ上で行うノンリニア編集が広く普及してきている．多チャンネル化と高精細度（HD）化が進む現在，いかに制作作業効率を向上させるかが特に重要な課題となっている．しかし，HD 画質での非圧縮ビデオ信号は膨大なデータ量となるため，ストレージの記録容量が足りない．そのため，編集段階での画質をサポートする MPEG2 を用いたノンリニア編集システムが必要とされている．

現在の放送番組制作におけるボトルネックは，制作が完了してから放送局に持ち込むまでに時間がかかってしまうことである．HD ノンリニア編集システムには，制作段階でのワークフローの向上のため，MPEG2 エンコードをできる限り短い時間で処理できる性能が求められている[13]．小高らによる OSCAR チップマルチプロセッサ上での MPEG2 エンコーディングの並列処理は，1 チップマルチプロセッサを用いたハードウェアによる高速化の手法であり，マクロブロックレベルでの並列化を行っている．プロセッサ数が 4 の場合，逐次実行時間に対して約 3．5 倍の速度向上が得られている[8][9]．

SMP クラスタ上でのハイブリッド並列プログラミングの性能は，LINPACK や NAS Parallel Benchmarks などの性能は公表されているが[7]，JEPG，MPEG に代表されるマルチメディア系アプリケーションにおける性能はあまり明らかになっていない．MPEG2 エンコーディングは，可変長符号化を行うため，そのままでは並列化は難しいが，ランダムアクセスの実現やエラー耐性のための GOP，スライス，マクロブロックの単位では独立

に処理ができるため，その性質を利用することで並列化が可能である．

本研究では，MPEG2 エンコーダをハイブリッド並列プログラミングと MPI による並列プログラミングの 2 通りで実現し，それらの性能を比較する．ハイブリッド並列プログラミングでは，各ノードへのデータ分割を MPI で行い，各ノード内では OpenMP を用いて 8×8 画素のブロックを均等に二分割して 2 プロセッサ並列実行を行う．OpenMP で並列化する処理は，処理量の多い DCT，IDCT と量子化，逆量子化である．一方，MPI による並列プログラミングでは，各ノード内にある 2 プロセッサをそれぞれ独立したノードとみなし，MPI で各ノードへのデータ分割を行う．それぞれの実行時間をノード台数を 1 から 16 まで変化させて計測し，ハイブリッド並列プログラミングと MPI による並列プログラミングの比較，及びブロック分割とサイクリック分割の比較を行った．

実験に用いるプログラムは，マルチメディア系ベンチマークソフトとしてよく使われる MediaBench にも収録されている MPEG Software Simulation Group によるオープンソースソフトウェアの mpeg2encode を用いる．入力動画には，デジタルビデオカメラで撮影，編集を行ったものを用い，Main Profile における Main Level と Low Level，High 1440 Level の 3 種類のデータサイズで実験を行った．

本論文では，2 章で SMP クラスタとハイブリッド並列プログラミングについて説明し，3 章では MPEG2 エンコーダのアルゴリズムを述べ，4 章ではハイブリッド並列プログラミングによる MPEG2 エンコーダの並列化の方法を説明し，5 章ではハイブリッド並列プログラミングによる実行結果を示し，その評価について述べる．

2. SMP クラスタとハイブリッド並列プログラミング

2. 1 SMP クラスタ

並列計算機は，図 1 のような共有メモリ型並列計算機と図 2 のような分散メモリ型並列計算機の 2 つに分類される．SMP クラスタは，2Way，4Way など複数のプロセッサを搭載した SMP 構成の PC をさらに複数台ネットワークで接続した並列計算機である．

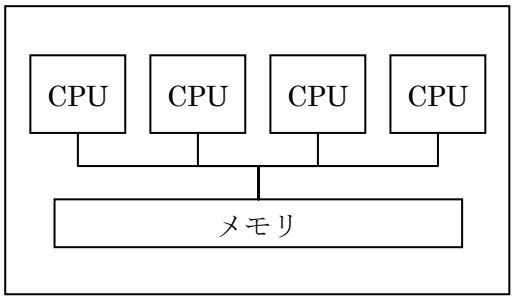

図 1：共有メモリ型並列計算機

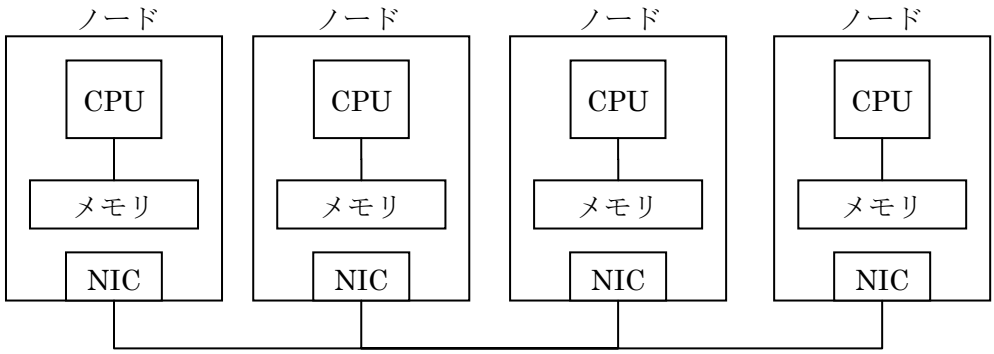

図 2：分散メモリ型並列計算機

SMP クラスタは，図 3 のように外観は分散メモリモデルと同じであるが，各ノード内は共有メモリモデルと同じ構成になっており，共有メモリモデルと分散メモリモデルとが混在したアーキテクチャである．一般的に PC クラスタのボトルネックはネットワークと言われており，プロセッサ数が同じであれば共有メモリモデルの方が性能面で優れている．SMP クラスタは，分散メモリモデルのように容易に演算ユニットを増やすことができ，ネットワークトラフィックを減らすことができるため，より高速な性能が見込める．

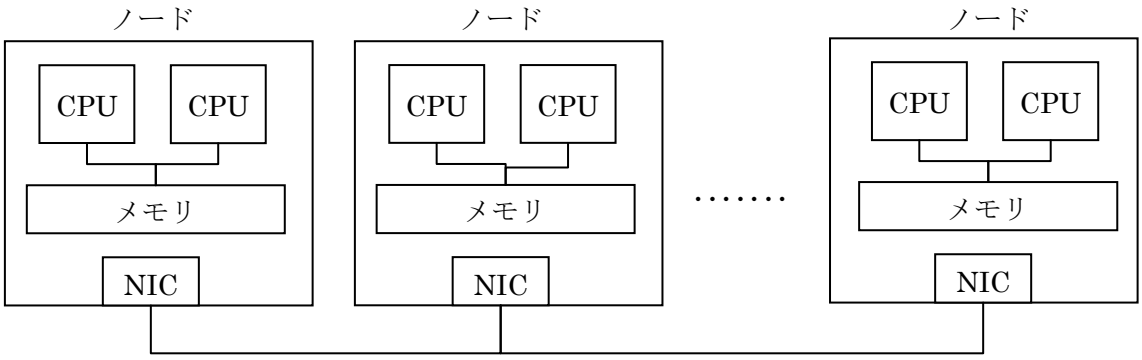

図 3：SMP クラスタ

2．2　ハイブリッド並列プログラミング

MPI は，ノード間の通信をメッセージの送受信によって実現するライブラリレベルでの並列化実装であるため，C や Fortrun など言語を問わず使用できる．共有メモリ環境におけるプログラミングには，以前より thread が使用されてきたが，thread はベンダー独自の実装である場合が多く，ソースコードレベルでの互換性もバイナリレベルでの互換性もない．最近になって POSIX スレッドとして標準化がされ，多くの UNIX 系 OS で普及しつつあるが，Windows や他の OS では未だサポートされていないのが現状である．一方，thread のインターフェースとして実装され，thread と比較して簡単にプログラミングが行える共有メモリプログラミングモデルの OpenMP が C や C++，Fortun などのライブラリとして実装され，多くのベンダー製コンパイラに採用されたために，普及が進んでいる．

ハイブリッド並列プログラミングとは，分散メモリ環境と共有メモリ環境が混在した環境において，分散メモリプログラミング環境と共有メモリプログラミング環境の両方の利点を生かしたプログラミングである．MPI は，ノード間での通信を行う並列化実装であるため，ノード内のプロセッサ間の通信を行う場合はオーバーヘッドが大きい．また，OpenMP は，ノード内のプロセッサでメモリ共有を行うためのプログラミングモデルであるため，分散メモリ環境で実現するにはノード間のメモリをソフトウェアで共有する必要があるためボトルネックとなる．また，全てのメモリを 1 つの共有メモリ空間として扱うため，メモリを有効に活用できない．よって，SMP クラスタ上ではノード間は MPI によって並列化し，ノード内は OpenMP によってメモリを共有するハイブリッド並列プログラミングを行うことが理想的である．図 4 にハイブリッド並列プログラミングにおける MPI と OpenMP の担当範囲を示す．

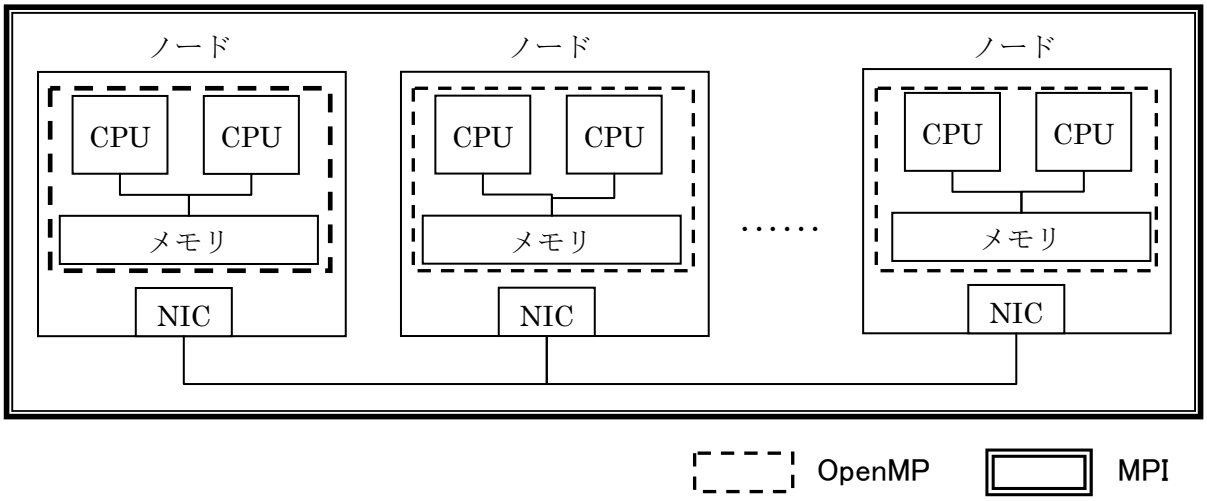

図 4：MPI と OpenMP の担当範囲

SMP クラスタ上での実際の実装は，MPI のみの並列化では，各ノードに対して MPI プロセスを 2 つ割り当て，全てのプロセッサに同じ処理を実行させている．それに対して，ハイブリッドによる並列化では，各ノードに対し MPI プロセスを 1 つずつ割り当て，ノード内では OpenMP スレッドを 2 つ生成して，各プロセッサに割り当て，並列動作を行っている．MPI による並列実行イメージを図 5 に，ハイブリッド並列プログラミングによる実行イメージを図 6 に示す.

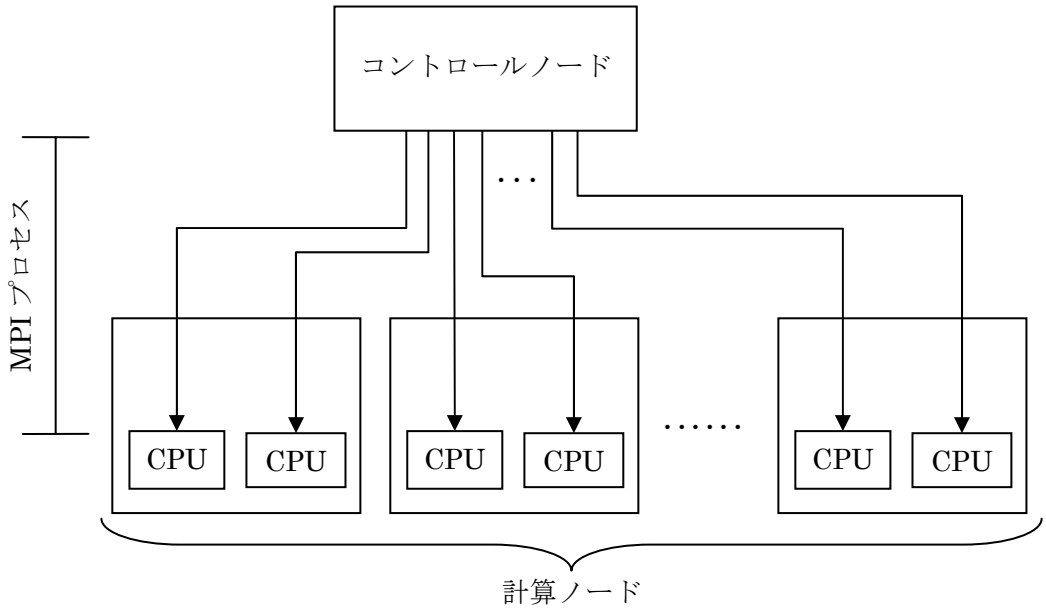

図 5：MPI による並列実行イメージ

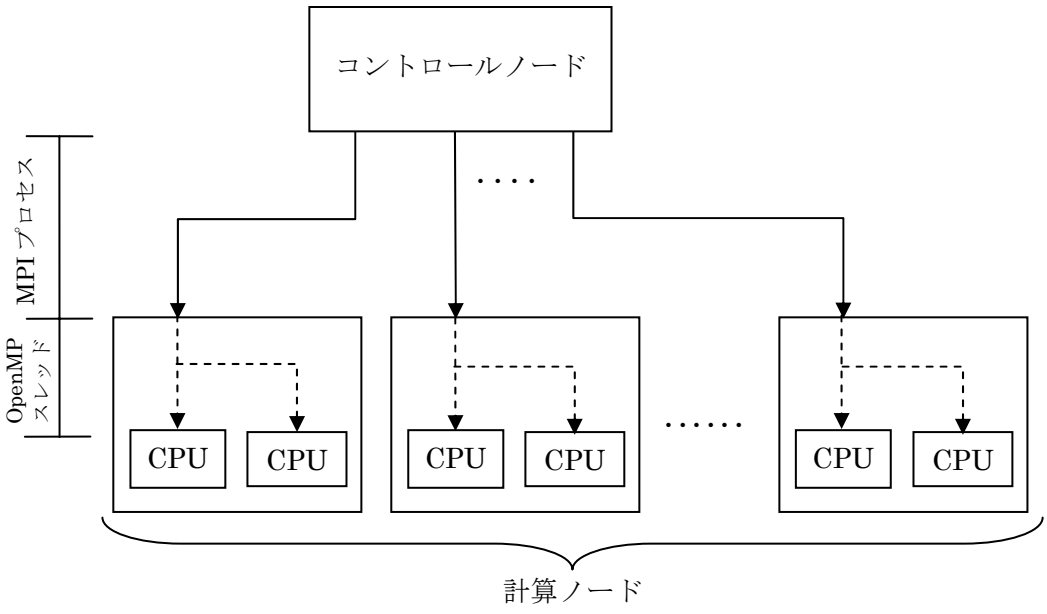

図 6 : ハイブリッド並列プログラミングによる実行イメージ

3. MPEG2 エンコーダのアルゴリズム[1][2]

3．1　概要

MPEG2 は，5～10Mbps 程度のビットレートで現行テレビ品質を実現する符号化方式として，Moving Picture Experts Group（MPEG）により標準化された．MPEG2 は MPEG1 と同様に，主にビデオ・音声・システムの 3 つの異なる標準からなっている．本論文では MPEG2 と表記した場合，ビデオの標準を取り扱うこととする．また，MPEG2 のアルゴリズムは複雑かつ膨大であるため，本章では並列化に関連の深い処理についてのみ記述する．

MPEG2 規格は，様々な目的に対応するために総合的な物となっており，全てを常時必須とすることは不合理である．そこで，Profile と呼ばれる技術の実施範囲規定を設けている．MPEG2 には現在 7 種類の Profile が存在するが，実際に使われているのは Main とより簡単な構成の Simple のみである．さらに処理量（パラメータ）の規定で Level と呼ばれる規定が設けられている．従って MPEG2 エンコーダやデコーダがサポートする技術範囲は Profile と Level の組み合わせで設定される．表 1 に Main Profile における各 Level のパラメータの最大値を示す．

表 1：Main Profile の各 Level の設定

パラメータ＼Level	High	High 1440	Main	Low
最大ビットレート(Mbps)	80	60	15	4
最大画素数/ライン	1920	1440	720	352
最大画素数/フレーム	1088	1088	576	288
最大フレーム数/秒	60	60	30	30

MPEG2 の Main Profile におけるアルゴリズムは，動きベクトルを使用したマクロブロック単位の動き補償方式を採用し，量子化された離散コサイン変換（DCT）係数の 2 次元符号化方式を採用している．また，現行のテレビを高画質なまま符号化するために，インターレースを直接符号化する方式が取られている．一方，データ構造は Group Of Picture（以下 GOP）と呼ばれる複数のピクチャからなる編集の単位を持ち，I・P・B ピクチャと呼ばれる符号化の分類を持つ．そして，各ピクチャはスライス，マクロブロック，ブロックという構造を持っている．図 7 に Main Profile における MPEG2 エンコーダのアルゴリズムの概要を示す．

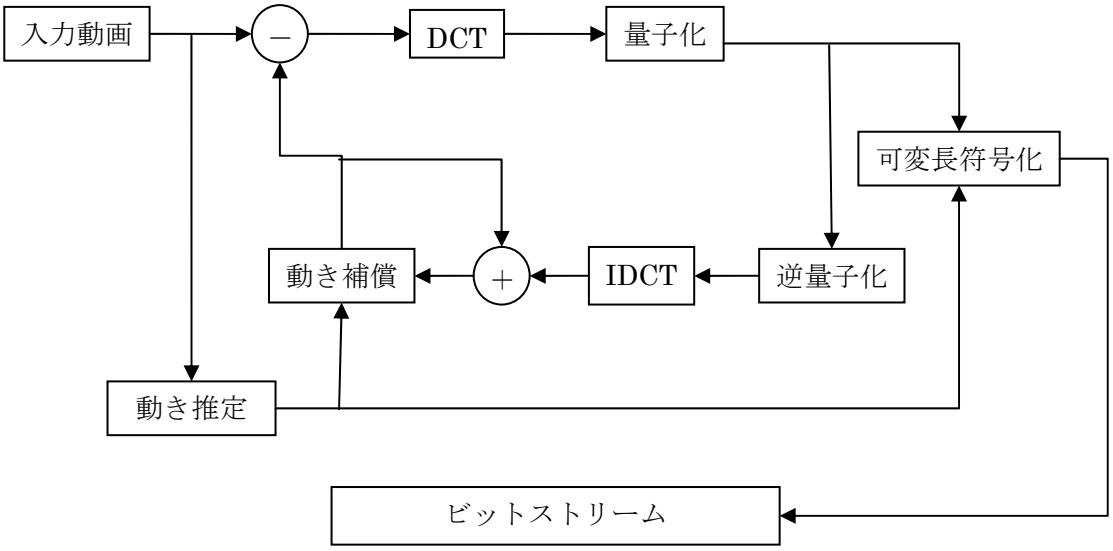

図 7：MPEG2 エンコーダのアルゴリズム

3．2　MPEG2 のデータ構造

図 8 に MPEG2 のデータ構造の階層構成を示す．MPEG2 のような蓄積メディア用画像符号化技術が，通信用のものと異なる最大の点は，時間軸の制約がないことである．すなわち，早送り，巻き戻し，途中からの再生が要求されることである．これらを満たすため，何枚かのピクチャをまとめて GOP（Group of Pictures）と呼び，その単位での独立再生ができるようになっており，GOP には途中からの再生を可能にするため，シーケンス・ヘッダを付けることができる．MPEG2 のデータ構造は，この GOP を頂点とした階層構造になっている．その構成要素は，上から GOP，ピクチャ，スライス，マクロブロック，そしてブロックとなっている．スライスは，任意の長さの 1 次元構造となっており，1 枚のピクチャは複数のスライスで構成されている．ブロックは，MPEG2 データの最小要素で，8×8 画素を 1 つの単位としている．DCT，IDCT や量子化，逆量子化はこのブロック単位で演算を行う．マクロブロックは，4 つの輝度ブロックと 2 つの色差ブロックである．予測や符号化モードの指定にはマクロブロック単位で行う．Main Profile における色差ブロックは，縦横に 1/2 にサブサンプリングされた縮小画像であるため，画面上の大きさは輝度の 4 ブロックと同じである．

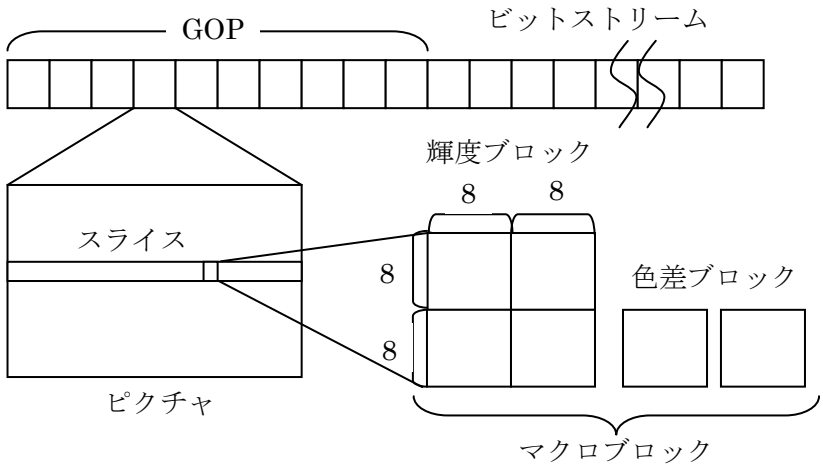

図 8：MPEG2 のデータ構造

3．3　フレーム間予測と動き補償

入力画像と以前の再生画像（エンコーダ内でデコードされた画像）との差分を符号化することを，フレーム間予測符号化と言う．符号化が誤差を許すため，入力画像同士の差でなく，以前の再生画像との差にすることにより，元々の画像がわからないデコーダが次の再生画像を作ることができる．エンコーダ内で再生画像を得るためには，エンコーダ内にローカルなデコーダを持ち，再生画像を作る必要がある．デコーダでは，符号から差分データに戻した後，以前の再生画像に対応する画素値に加算して新しい再生画像を作る．

基本的に，動画像の大半の領域は以前の画像と近似し，フレーム間予測の差分の情報量は，一般に入力画像の情報量より小さいため，フレーム間予測は符号化効率を高める．さらに，画像をブロックに分割し，ブロックごとに差分の符号化と，入力画像自身の符号化の選択を可能にして，差分が大きいブロックには入力画像自身を使うこともできる．切り替え情報はブロックの付加情報に付け加える．図 9 にフレーム間予測の処理手順を示す．

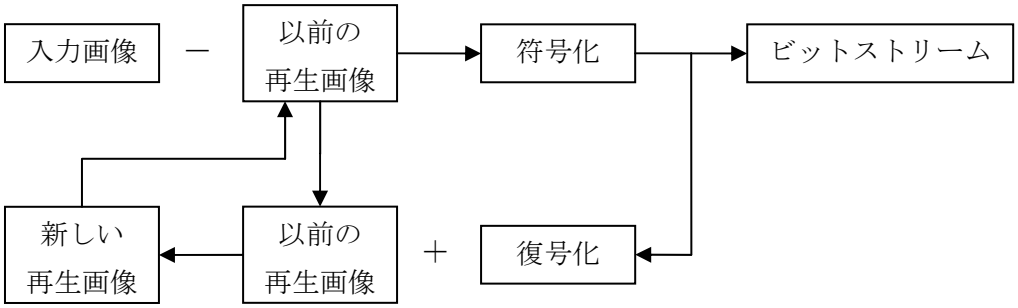

図 9：フレーム間予測の処理手順

フレーム間予測は，次の 3 種類のピクチャタイプに分類される．

・ Ｉピクチャ（INTRA）

フレーム間予測を行わず，フレーム内符号化を行う．Ｉピクチャの目的は，GOP の独立性を保つことである．

・ Ｐピクチャ（INTER）

フレーム間予測を行うか，フレーム内符号化かを選択することができる．フレーム間予測は過去（の再生画像）から予測を行う．

・ Ｂピクチャ（INTER）

双方向の予測を行う．過去の I，P ピクチャだけでなく，未来の I，P ピクチャをも予測に使う画像であり，P ピクチャに比べてさらに情報量を小さくすることができる．未来の画像を予測に使う場合には，未来に使う画像を先に入力し，後から B ピクチャを入力する．よって，原画像と出力画像の順番は入れ替わることがある．

図 10 にフレーム間予測におけるピクチャタイプの関係を示す．

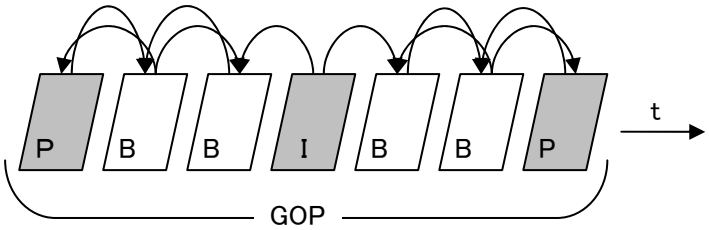

図 10：フレーム間予測とピクチャタイプ

さらに，各ブロックの参照画像上の対応位置からのずれ量を示す「動きベクトル」を用意し，参照画像上の動きベクトル分だけずらした場所から予測するフレーム間予測を，「動き補償」と言う．

動きベクトルの符号量は，動き補償による予測誤差の符号量減少より小さく，滑らかに変化する動画像では，動き補償の予測誤差の情報量を減少させる効果は，単純なフレーム間予測よりはるかに大きく，符号化を効率的にする．

動きベクトルとは，下式の（mvx,mvy）であり，現在のブロックの位置に加算して参照画像上の位置とする．

$$\mathrm{X}_{i,j,k} = \begin{cases} I_{i,j,k} - R_{i+mvx,j+mvy,k-1}(INTER) \\ I_{i,j,k}(INTRA) \end{cases}$$

ここで，i,j を画素の画像上の位置，k を画像の番号とし，$X_{i,j,k}$ は差分符号化画像，$I_{i,j,k}$ は入力画像，$\mathrm{R}_{i,j,k-1}$ は再生画像とする．

3．4　DCT と IDCT

フレーム間動き補償予測誤差のマクロブロックの 6 つのブロックに対して，8×8 の 2 次元 DCT を行う．DCT は画像信号を少数の低域係数に集中させる働きを持ち，画像の空間的な情報量削減に使われる．DCT は固定の変換係数の直交変換の中で，画像符号化に最も有効な変換とされる．

N 点 1 次元 DCT と，その逆変換（IDCT）は，次のように定義される．

x_i を入力とし，X_k をその DCT 変換係数とするとき，

$$X_k = \sqrt{\frac{2}{N}} C(k) \sum_{i=0}^{N-1} x_i \cos\left[\frac{\pi k(2i+1)}{2N}\right]$$

$$x_i = \sqrt{\frac{2}{N}} \sum_{k=o}^{N-1} C(k) X_k \cos\left[\frac{\pi k(2i+1)}{2N}\right]$$

ただし，$C(n) = \begin{cases} \dfrac{1}{\sqrt{2}} (n=0) \\ 1 \quad (n \neq 0) \end{cases}$

1 次元 DCT は，8 つの DCT 係数を求めるために 64 回の積和処理が必要である．

N×N の 2 次元 DCT と IDCT は次の通りである．

$$X_{k,l} = \frac{2}{N} C(k)C(l) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x_{i,j} \cos\left[\frac{\pi k(2i+1)}{2N}\right] \cos\left[\frac{\pi l(2j+1)}{2N}\right]$$

$$x_{i,j} = \frac{2}{N} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} C(k)C(l) x_{i,j} \cos\left[\frac{\pi k(2i+1)}{2N}\right] \cos\left[\frac{\pi l(2j+1)}{2N}\right]$$

2 次元 DCT の処理は 1 次元 DCT の処理に次元分解でき，計算量も少なくなる．これは，ブロック内の各行を x 方向に 1 次元 DCT し，結果を同じ行に戻し，つぎに y 方向に 1 次元 DCT を各列で行い，結果を同じ列に戻すことである．次元分解を行った 2 次元 DCT と IDCT は次の通りである．

$$g_{k,j} = \sqrt{\frac{2}{N}} C(k) \sum_{i=0}^{N-1} x_{i,j} \cos\left[\frac{\pi k(2i+1)}{2N}\right]$$

$$X_{k,j} = \sqrt{\frac{2}{N}} C(l) \sum_{k=o}^{N-1} g_{k,j} \cos\left[\frac{\pi l(2j+1)}{2N}\right]$$

式通りの 8×8 の 2 次元 DCT は，64×64 回の積和であるが，次元分解すると，16 回と 1/4 になる．

3．5　量子化と逆量子化

MPEG2 規格では，逆量子化式は決まっているが，量子化式（量子化マトリクス）はエンコーダで自由に設定できる．イントラの DC 係数以外では，輝度，色差とも次式で行われる．逆量子化された再現値を F，量子化マトリクスを QM，量子化された DCT 係数を W とした時，逆量子化式は次式の通りである．

$$F[v][u] = \{(2QM[v][u] + k) \times W[u][v] \times quantizer\_scale\} / 32$$

量子化スケールはスライスヘッダで設定され，マクロブロック単位で変更できる量子化のステップである．デフォルトの量子化マトリクスを図 11 に示す．

8	16	19	22	26	27	29	34
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	40	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	59	83

Intra

16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16

Non-Intra

図 11：デフォルトの量子化マトリクス

4. 並列化アルゴリズム

4. 1　分散メモリプログラミングによる並列化

MPEG2 では，フレーム間予測を行うため，任意のフレームで入力データを分割した並列化はできない．しかし GOP にはフレーム間予測を行わない I ピクチャが含まれており，フレーム間予測は GOP 内で完結している．よって，GOP と GOP の境界でのデータ分割は可能であり，並列化が可能である．

SMP クラスタでは，1 ノード内に複数のプロセッサが搭載されているが，分散メモリプログラミングで並列化した場合は各プロセッサがノードとして扱われる．1 ノード当たり 2 プロセッサ，16 ノードの SMP クラスタの場合は，32 ノードのクラスタとして動作する．

分散メモリプログラミングによる並列化は，ブロック分割，サイクリック分割の 2 通りの並列化手法で行った．

（1）ブロック分割

ブロック分割は，入力動画を単純にノードの台数で分割して並列化する．図 12 にブロック分割による並列化手法を示す．

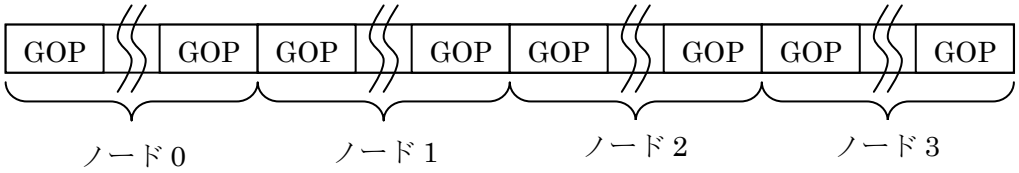

図 12：ブロック分割

ブロック分割は，並列化によるサーバーとノードとの通信回数が 1 度で済むため，全ての処理範囲が同じ処理量であれば最も効率が良い．

（2）サイクリック分割

サイクリック分割は，入力動画を小さい単位で区切り，順番にプロセッサに割り当てて並列化を行う．図 13 にサイクリック分割による並列化手法を示す．

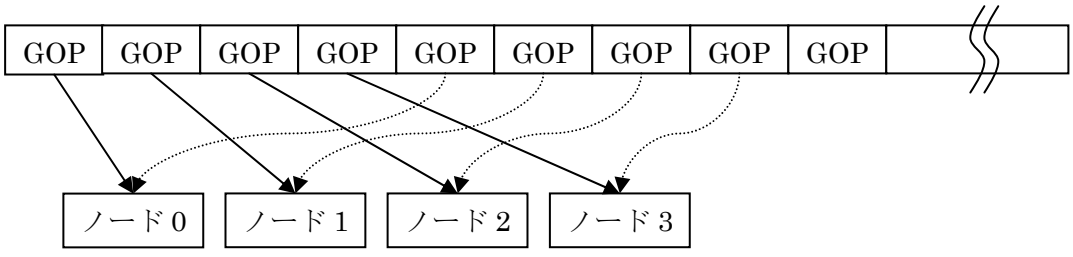

図 13：サイクリック分割

サイクリック分割は，サーバーとノードとの通信回数が多くなってしまうが，処理量を分散することができ，処理範囲毎に計算量が違う場合に効率が良くなる．動画像を扱う場合，通常同じ画像が入力されることはないため，GOP 毎に処理量が変化する．そのため，

MPEG2 エンコーダを並列化する場合は, ブロック分割よりもサイクリック分割の方が負荷を分散することができ，効率がよいと考えられる．

4．2　共有メモリプログラミングによる並列化[3]

共有メモリプログラミングでは，分散メモリプログラミングとは異なり，演算ブロックでの並列化を行う．実際に並列化を行ったのは処理量の多い DCT，IDCT と量子化，逆量子化である．

DCT と IDCT は，1 つの画素を変換するためにブロック内の全ての値が必要である．また，ブロックの左上の画素は隣のブロックの差分値を符号化する．よって，ブロックを分割することはできず，ブロック毎に分割することもできない．分散メモリプログラミングを用いてブロック内で並列化する場合は，プロセッサがそれぞれの担当範囲を処理するには全てのデータを受信している必要があり, 1 つのブロックの処理が終わらなければ次のブロックでの差分値を計算することができないため，ブロック外で並列化することはできない．しかし，共有メモリプログラミングで並列化する場合は，それぞれのプロセッサが同じメモリを共有し，ブロックの全てのデータを参照することができるため，それぞれの担当範囲を計算する際に余分なデータ転送などは発生しない．並列化アルゴリズムには，分散メモリプログラミングによる並列化と同様にブロック分割を用いるが，ここでは 1 枚のピクチャ内のブロックをプロセッサ数で均等に分割する．

量子化と逆量子化は，処理画素毎に独立して処理が行える．よって，ブロックをプロセッサ数で均等に分割し，DCT と IDCT と同じ方法で並列化を行った．

DCT，IDCT と量子化，逆量子化の並列化の方法を図 14 に示す．

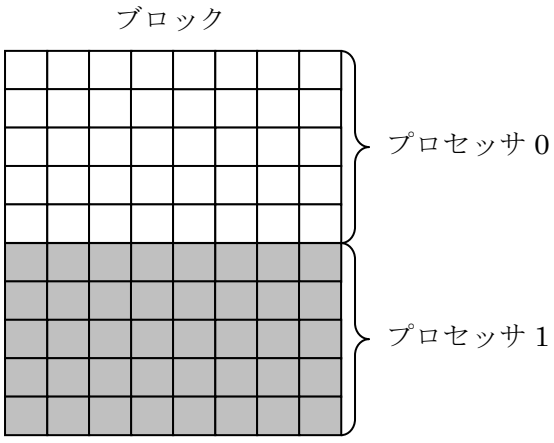

図 14：共有メモリ環境における並列化

4．3　ハイブリッド並列プログラミングによる並列化

本論文で評価を行うハイブリッド並列プログラミングは，上記 2 種類のメモリ環境での並列化手法を組み合わせ, MPI により各ノードに GOP を単位とした数フレームを割り当てて並列化し，さらに DCT，IDCT と量子化，逆量子化では OpenMP により並列化する．図 15 にハイブリッド並列プログラミングによる並列化の方法を示す.

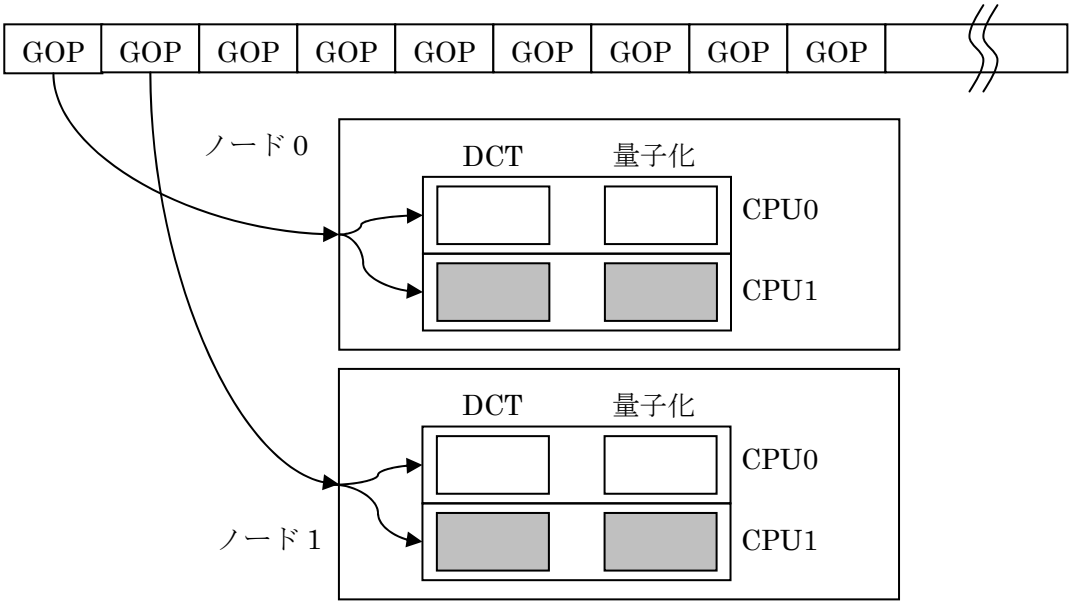

図 15：ハイブリッド並列プログラミングによる並列化

この手法では，DCT，IDCT と量子化，逆量子化以外の処理はノード毎に 1 プロセッサで行う.

5. 実験と考察

5. 1 実験環境

実験環境には，Intel Xeon2.8GHz を 2 台搭載した SMP ノード 16 台を Gigabit Ether で接続した，計 32 プロセッサの SMP クラスタ「Atlantis」を用いる．図 16 にその構成を示す.

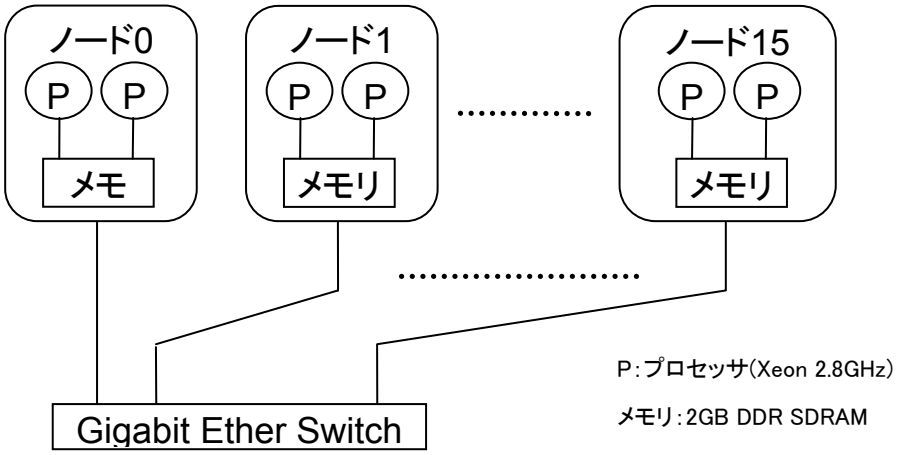

図 16：SMP クラスタ Atlantis の構成

Atlantis の各ノードは，CPU に Intel Xeon プロセッサ 2.8GHz を 2 台，メモリに 2GB の DDR SDRAM を搭載した SMP 構成となっている．各ノードは Gigabit Ether で接続されており，クラスタシステムソフトウェア SCore を用いて PC クラスタとして構築されている．MPI コンパイラには，SCore 版 MPICH version 1.2.5 を用い，C コンパイラ及び OpenMP コンパイラには Intel C++ Compiler version 9.0 を用いた.

5. 2 実験条件

使用する Profile は Main Profile で，Main Level，Low Level，High 1440 Level の 3 種類の実験を行った． Main Level における入力動画は，図 17 のような 720×480 画素の自然動画で，Low Level，High 1440 Level の入力動画は，これにリサンプリングを行った動画を用いた．また，動画の長さは 64 秒間，1920 フレームである［6］.

入力動画は，Portable Pix Map（PPM）形式の連続した静止画像であり，1 フレーム当りのデータサイズは Main Profile で 1MB, Low Level で 0.3MB, High 1440 Level で 4MB である．また，ノード数ごとの 1 ノード当りのフレーム数とデータサイズを表 2 に示す.

図 17：入力動画

表 2：ノード当りのデータサイズ（単位：MB）

ノード数	1	2	4	8	16
ノード当りのフレーム数	1920	960	480	240	120
Main Level	1920	960	480	240	120
Low Level	576	288	144	72	36
High Level	7680	3840	1920	960	480

5．3　実験結果

エンコード後のファイル容量は，Main Level で約 120MB，Low Level で約 30MB，High 1440 Level で約 640MB で，5～10%の圧縮率である．

（1）実行時間

実行時間を表 3 に示す．表中の MP@ML は Main Profile の Main Level を示し，MP@LL は Low Level を示し，HL は High 1440 Level を示す．また，Main Level での実行時間のグラフを図 18 に，Low Level での実行時間のグラフを図 19，High 1440 Level での実行時間のグラフを図 20 に示す．1 ノード当たりのプロセッサ数は 2 台で，16 ノードでは 32 プロセッサ全てを使った実行結果である．

表 3：実行時間（単位：秒）

	ノード数	1	2	4	8	16
MP@ML	ブロック分割(ハイブリッド)	1248.5	629.3	322.5	184.9	158.0
	ブロック分割(MPI)	1428.6	721.6	389.9	234.7	159.1
	サイクリック分割(ハイブリッド)	1314.7	661.9	338.4	187.3	150.3
	サイクリック分割(MPI)	1501.1	762.9	397.0	247.7	154.3
MP@LL	ブロック分割(ハイブリッド)	291.4	150.5	80.1	62.8	70.9
	ブロック分割(MPI)	349.7	177.8	93.5	65.2	68.2
	サイクリック分割(ハイブリッド)	315.8	158.9	85.6	60.1	63.9
	サイクリック分割(MPI)	372.0	183.1	97.5	63.4	63.7
MP@HL	ブロック分割(ハイブリッド)	6014.4	3094.3	1543.0	795.9	444.2
	ブロック分割(MPI)	7014.9	3551.5	1791.3	905.7	508.5
	サイクリック分割(ハイブリッド)	6344.9	3247.2	1611.7	819.1	458.5
	サイクリック分割(MPI)	7137.1	3673.9	1863.7	961.1	514.5

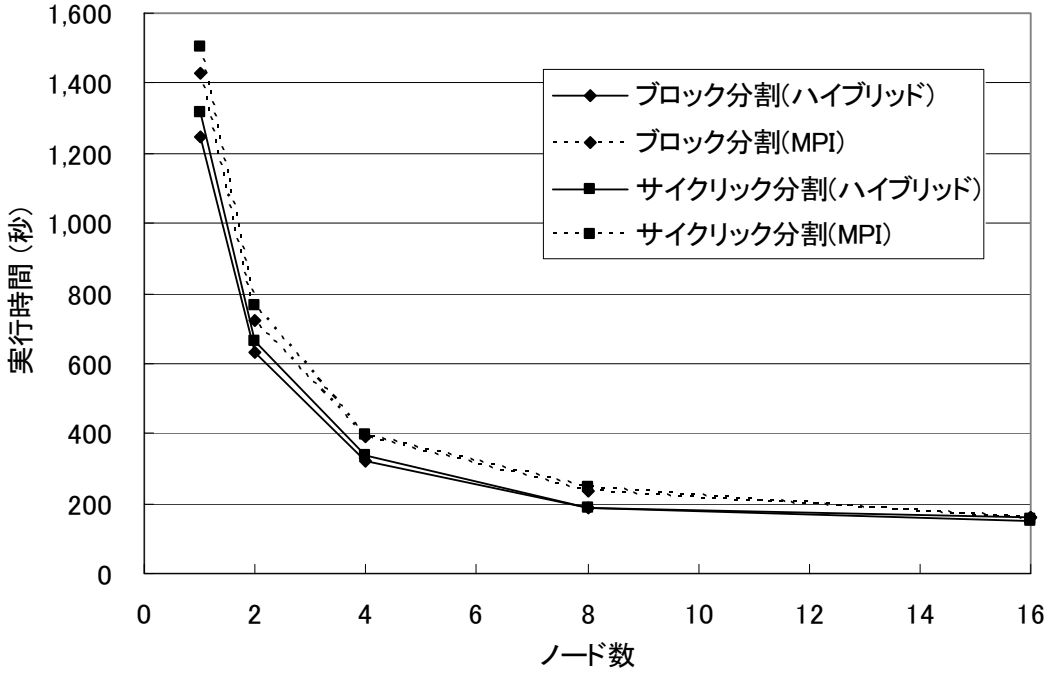

図 18：Main Profile，Main Level での実行時間

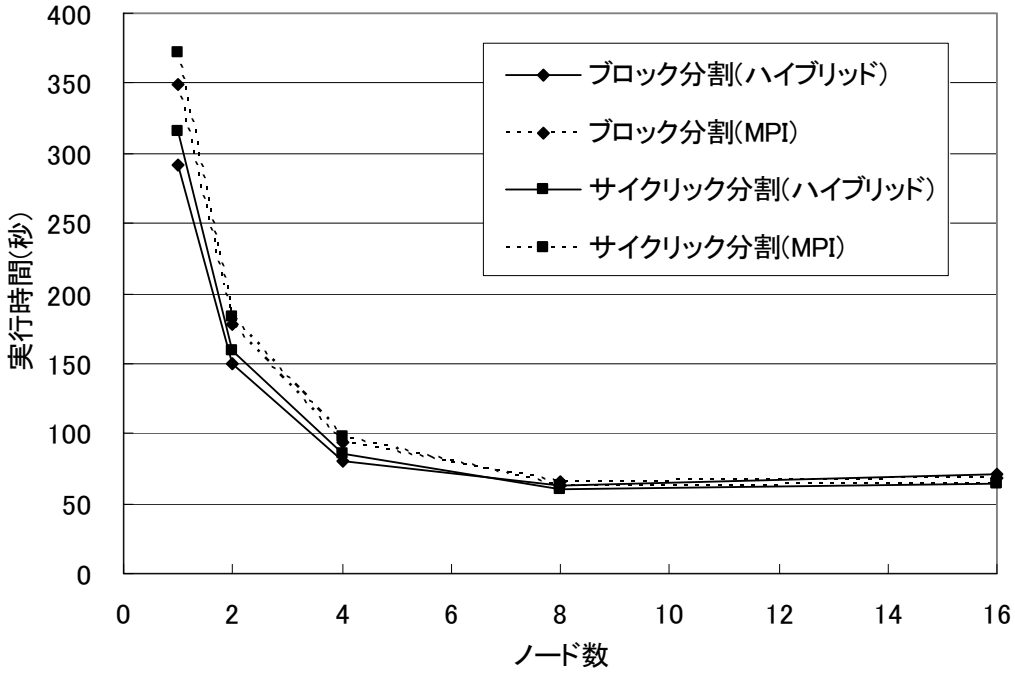

図 19：Main Profile，Low Level での実行時間

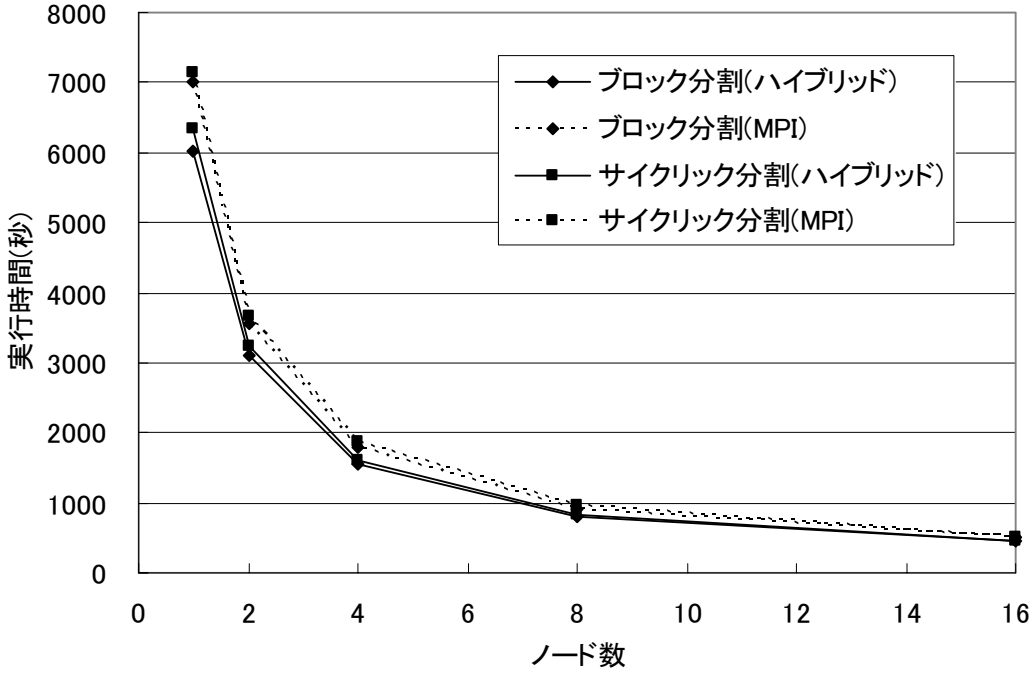

図 20：Main Profile，High 1440 Level での実行時間

16 ノードにおける Low Level の結果を除いてハイブリッドでの結果の方が MPI のみの場合に比べて常に実行時間は短いということが分かる．特に，1 ノードでの実行時間は，ハイブリッドにより大きく短縮されている．また，ブロック分割での結果とサイクリック分割での結果を比較すると，1 ノードでの実行時間はブロック分割の方が短いが，16 ノードではサイクリック分割での結果の方が実行時間は短くなっている．

ここで，ハイブリッドによる実行時間の短縮率を表 4 に示す．

表 4：実行時間の短縮率（単位：%）

Profile@Level＼ノード数	1	2	4	8	16
MP@ML(サイクリック分割)	12.41	13.24	14.75	24.38	2.62
MP@LL(サイクリック分割)	15.11	13.22	12.18	5.16	−0.22
MP@HL(サイクリック分割)	11.10	11.62	13.52	14.77	10.88
MP@ML(ブロック分割)	12.61	12.78	17.29	21.23	0.71
MP@LL(ブロック分割)	16.68	15.40	14.29	3.73	−3.90
MP@HL(ブロック分割)	14.26	12.87	13.86	12.13	12.64

ほとんどの場合 10%～20%の短縮率となっており，サイクリック分割における Main Level，8 ノードでの 24.38%が最も短縮率が良い．しかし，Low Level における 8 ノードでの実行時間の差が小さくなり，16 ノードでは逆に実行時間が長くなっている．また，Main Level における 16 ノードでの実行時間の差も小さくなっている．

（２）速度向上

Main Level での速度向上を図 21，Low Level での速度向上を図 22，High 1440 Level での速度向上を図 23 示す．

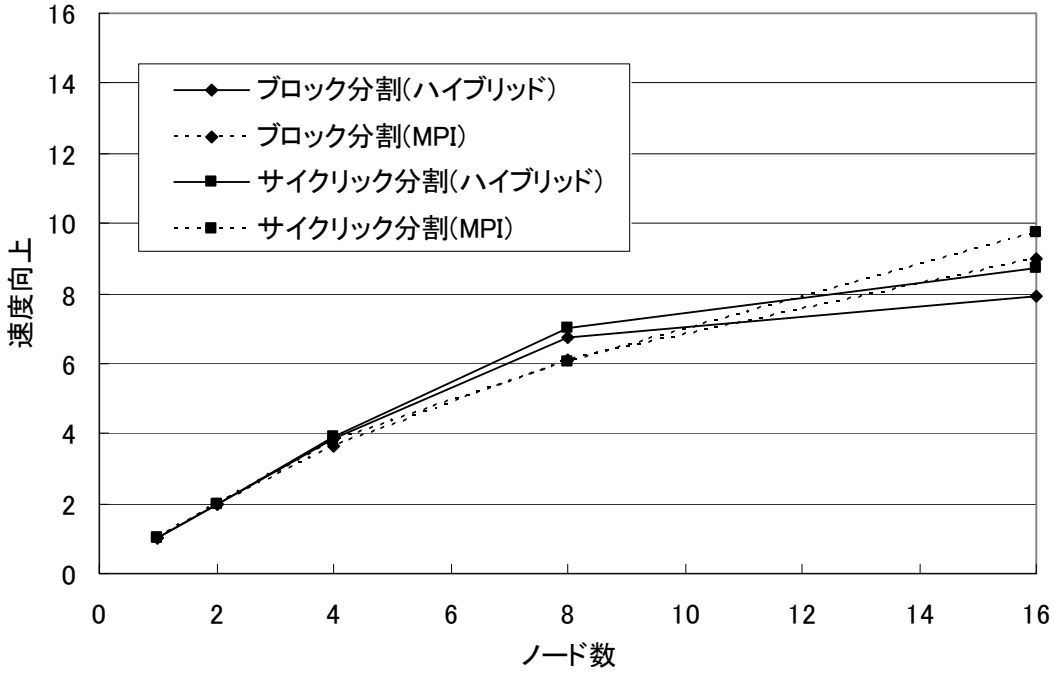

図 21：Main Profile，Main Level での速度向上

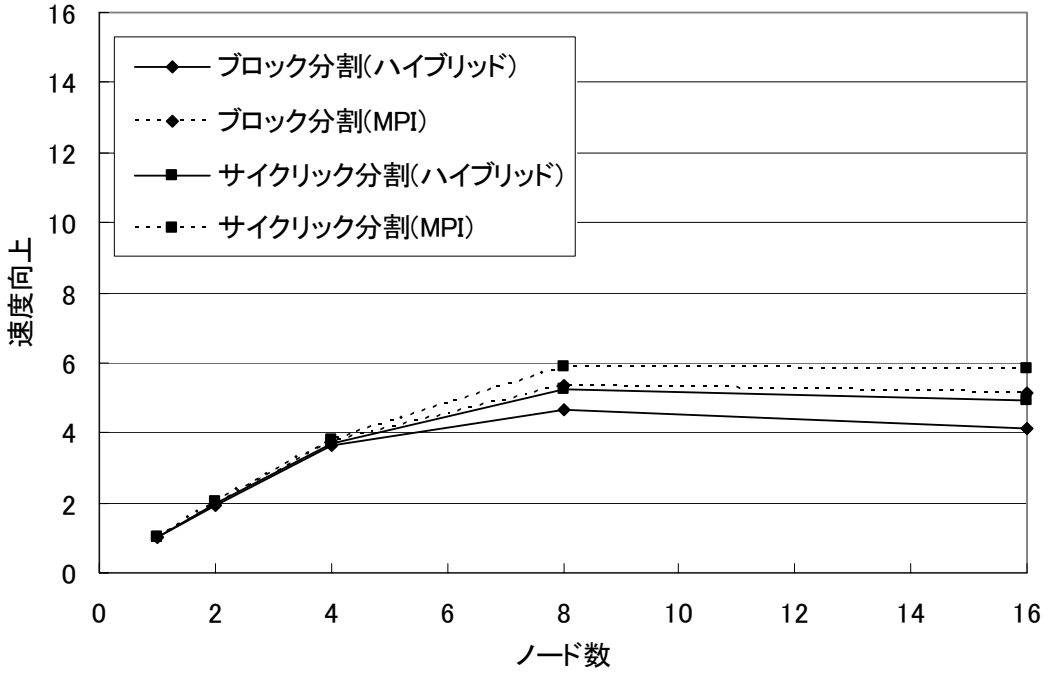

図 22：Main Profile，Low Level での速度向上

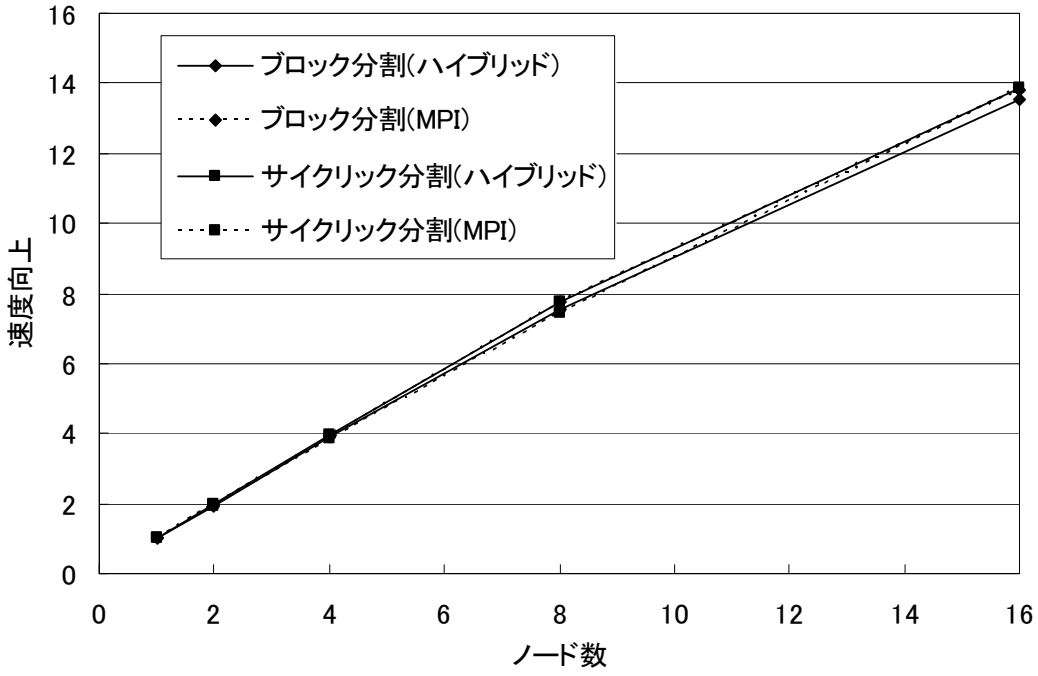

図 23：Main Profile，High 1440 Level での速度向上

速度向上はブロック分割による結果の方が高く，Main Level では 16 ノードで約 9.7 倍，Low Level では 8 ノードで約 5.8 倍，High 1440 Level では 16 ノードで約 13.9 倍となった．Low Level では，8 ノードでの速度向上とほぼ同じ速度向上であった．

5．4　考察

（1）ハイブリッド並列プログラミング

MPEG2 エンコーダは，ハイブリッド並列プログラミングにより MPI のみの場合と比べ，10%～20%実行時間を短縮することができ，OpenMP を併用した方が高速であることが確認できた．高い並列化効果が得られることで有名な多くの問題において，MPI と OpenMP を組み合わせたハイブリッド並列プログラミングを用いた場合，あまり性能が良くならないか，逆に下がってしまうことが多い[7]．今回の結果は，共有メモリ並列プログラミングと DCT，IDCT の相性が良く，理想的な速度向上が得られるため，MPI のみの並列化よりもハイブリッドでの実行結果の方が良くなったと考えられる．

表 4 より，Main Level では 16 ノードから，Low Level では 8 ノードからハイブリッドによる効率が低くなっている．表 2 と照らし合わせると 16 ノードでの Main Level では，ノード当りのデータサイズが 120MB で，このデータサイズ以下の時にハイブリッドによる効果の低下が見られるようになる．16 ノードでの Low Level では，36MB になり，MPI のみでの実行時間よりもハイブリッドでの実行時間の方が長くなっている．並列化効果は，

データサイズとのトレードオフとなるため，ハイブリッドによる十分な効果が出るボーダーラインは 120MB あたりであると考えられる．

（２）ブロック分割とサイクリック分割

ブロック分割とサイクリック分割を比較すると，サイクリック分割の方が 10%程度高速であることがわかる．また，2 ノードまではブロック分割の方が実行時間は短いが，それ以上ではサイクリック分割が逆転する．サイクリック分割では，実行するノード数に関わらず MPI による通信の頻度は一定である．つまり 1 ノードで実行する場合でも総フレーム数／GOP 回の通信が行われる．それに対し，ブロック分割ではノード数に関わらず始まりの 1 度だけであるため，2 ノード程度での実行時間はブロック分割の方が短い．

（３）速度向上

速度向上は，MPI によるサイクリック分割での結果が最も良い．Low Level では 8 ノードで 5．8 倍，Main Level では 16 ノードで 9．7 倍，High 1440 Level では 13．7 倍の速度向上となっている．Low Level では 2 ノードまでは理想的な速度向上が得られ，Main Level，High 1440 Level では 8 ノードまでは理想的な速度向上が得られたことより，データサイズが大きいほど理想的な速度向上に近づくと考えられる．

（４）データの分割単位

更なる検討課題として，エンコーダの更なる高速化が挙げられる．本論文のサイクリック分割では，分割の最小単位を GOP としており，15 フレームとしている．しかし，この値は可変であり，総フレーム数が増えた場合は数十～100 フレーム程度まで増やすことがある．このような場合において分割の最小単位を 2～10 程度の複数の GOP にするなど，1 ノードに与えるデータサイズを変化させ，最大ノード数に対する最適な分割方法について検証を行わなければならない．

（５）逐次プログラムの高速化

実験に用いたプログラムはリファレンス実装であり，cos 計算に math.h のマクローリン展開を使った関数を用いていることからもエンコーダ自身には何ら高速化アルゴリズムも採用されていない．よって，1 ノードでのエンコード時間は Main Profile，Main Level で入力動画の時間の 200 倍もの時間を要しており，16 ノードでも入力動画の時間の 2．5 倍のエンコード時間を要している．ここに表引きによる cos 計算などの高速化アルゴリズムを採用すれば，さらなる高速化が可能である．

6. おわりに

本研究では，SMP クラスタを有効に利用できるハイブリッド並列プログラミングを用いた MPEG2 エンコーダの並列化手法について検討，実装を行い，2Way16 ノードの SMP クラスタを用いて評価を行った．

ノード間は MPI を用いて並列化を行い，各ノードに均等に入力データを割り当てるブロック分割と入力データを GOP 毎に各ノードに割り振るサイクリック分割の 2 通りの並列化アルゴリズムを実装した. また, 各ノード内は OpenMP を用いて並列化を行い, DCT, IDCT と量子化，逆量子化について処理単位となる 8×8 画素のブロックを均等に分割し，各プロセッサに割り当てるブロック分割を用いて実装した．

ハイブリッド並列プログラミングにより，ブロック分割，サイクリック分割ともに MPI のみの結果に比べ 10～20%実行時間が短縮された．また，1 ノード当りのデータサイズが 120MB 以下の時ハイブリッド並列プログラミングによる効果が小さくなることを確認した. ブロック分割とサイクリック分割を比較すると，2 ノードまではブロック分割による並列化の方が実行時間は短いが，4 ノード以上ではサイクリック分割の方が 10%程度高速であるという結果が得られた. そして, Main Profile における各 Level の速度向上の最大は, Main Level で 9.7 倍（16 ノード），Low Level で 5.8 倍（8 ノード），High 1440 Level では 13.7 倍（16 ノード）の結果が得られた．

今後の研究課題として，第 5 章で述べた事項を検証するとともに，MPEG2 エンコーダの高速化だけでなく，他の圧縮コーデックやレンダリングに時間のかかるトランジションやエフェクト，フィルタなども合わせて高速化し，トータルとして快適なビデオ制作環境を構築することなどが挙げられる．

謝辞

先ず，本研究の機会を与えてくださり，貴重な助言，ご指導をいただきました山崎勝弘教授に深く感謝致します．また，コンピュータシステム研究室の PC クラスタ Atlantis の利用を快諾して下さった小柳滋教授に深く感謝致します．

そして，メーリングリストでの質問に関して，貴重な助言を頂きました PC Cluster Consortium の亀山豊久氏，筑波大学の佐藤三久教授に深く感謝致します．

さらに，本研究にあたり，貴重なご意見をいただきました高性能計算研究室の皆様にも心より感謝致します．

最後に，本研究の共同研究者である加藤氏に心より感謝致します．

参考文献

[1] 藤原洋，安田浩: “ポイント図解式 ブロードバンド+モバイル 標準 MPEG 教科書”，アスキー，2003.

[2]半谷精一郎，杉山賢二，“JPEG-MPEG 完全理解”，コロナ社，2005.

[3]加藤寛暁: “SMP クラスタ上での MPEG2 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，2006.

[4]PC Cluster Consortium

http://www.pccluster.org/.

[5]池上広済他: “PC クラスタ上での JPEG エンコーダ・デコーダの並列化”，FIT2005，No.A-030，pp．69-72，2005.

[6]池上広済，松村敏幸: “AWAKE”，http://www.cool-creators.com/，2002.

[7] 吉川茂洋他: “SMP-PC クラスタにおける OpenMP+MPI の性能評価”，情報処理学会研究報告，Vol.2000，No.023，pp．155-160，2000.

[8] 小高剛他:“OSCAR チップマルチプロセッサ上での MPEG2 エンコーディングの並列処理”，情報処理学会研究報告，Vol,2003，No.084，pp.55-60，2003.

[9]小高剛他: “チップマルチプロセッサ上での MPEG2 エンコードの並列処理”，情報処理学会論文誌，Vol.46，No.9，pp.2311-2325，2005

[10] Ta Quoc Viet: “Construction of Hybrid MPI-OpenMP Solutions for SMP Clusters”，情報処理学会論文誌，Vol.46，No.SIG3，pp.25-37，2005.

[11] Ta Quoc Viet: “MPI-OpenMP ハイブリッド分散並列プログラミングとそのアルゴリズムに関する研究”，電気通信大学大学院修士論文，2004.

[12] Ta Quoc Viet: “SMP クラスタにおけるハイブリッド MPI-OpenMP プログラミングのためのマスタースレーブアルゴリズム”，情報処理学会研究報告，2002.

[13] 清水文雄他:”小特集 最近の放送番組制作技術”，映像情報メディア学会誌，Vol.56，No.10，pp.1542-1548，2002.

[14]藤原洋: “ポイント図解式 実践 MPEG 教科書”，アスキー出版局，1995.

[15]大久保榮，角野眞也，菊池義浩，鈴木輝彦，“H.264/AVC 教科書”インプレス，2004.

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

[17]石川祐，佐藤三久，堀敦史，住元信司，原田浩，高橋俊行: “Linux で並列処理をしよう-SCore で作るスーパーコンピュータ”，共立出版社，2002.

[18] P．パチェコ: “MPI 並列プログラミング”，培風館，2001.