卒業論文

SMPクラスタ上でのOpenMPによるMPEG2エンコーダの並列化

氏名　　：　加藤　寛暁

学籍番号　：　2210020124-4

指導教員　：　山崎　勝弘　教授

提出日　　：　2006年2月20日

立命館大学　理工学部　情報学科

内容梗概

並列処理を行うことによって，大規模な計算の時間短縮が可能となる．近年，PC の高性能化に伴い，それまで大規模な計算機でしかできなかったことが，個人レベルでの PC で実現できるようになった．その中で代表的なものが動画編集である．また，PC を使ってのMPEG2 編集が一般的になってきた．しかし MPEG 2 編集においてネックになるのがエンコード時間である．動画によっては何時間もかかる場合がある．OpenMP の並列化プログラミングによるエンコード時間の高速化が本研究の目的である．

本論文では，OpenMP と MPI によるハイブリッドプログラム作成の共同研究として，OpenMP による並列化を担当しノード内並列化を行った．実験環境は OpenMP で並列化したプログラムは Xeon2.8GHz, 2CPU の共有メモリ環境「Tyrano」で行い，ハイブリッド並列プログラミング並列化では，1 ノード 2 プロセッサで 16 ノードの SMP クラスタ「Atrantis」で実験を行った．

64 秒間の画像を対象に， OpenMP で DCT, IDCT, 量子化, 逆量子化を並列化し，2CPU で約 1.9 倍の速度向上が得られた．

目次

図目次

表目次

1.はじめに

近年，汎用PCの高性能化に伴い，マルチプロセッサ･デュアルコアプロセッサを搭載したPCが一般的になりつつある．これらのPCでは，CPUをうまく利用すれば，１台のPCでも，シングルプロセッサ・シングルコアのプロセッサのマシンと比較して，様々な処理での高速化が実現できる．本研究では，OpenMP＋MPIを用いたハイブリッド並列プログラミングのOpenMP部分を担当すると共に，１PCマルチプロセッサ環境での速度向上の有効性の検証を行った.[5]

ハイブリッド並列プログラミングでは，SMPクラスタを用いた性能評価を行う．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．今日，並列プログラミングの主流になっているのがMPIである．MPIではSMPノード内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OpenMPによって分散共有メモリモデルにより並列化を行った場合には，ノード間のメモリの同期をソフトウェアで処理するため，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で通信して，ノード内ではOpenMPによりデータ共有を行うことでSMPクラスタ上で理想的な並列化が可能になる．

一方，コンピュータやハードディスクの低価格化，高機能化に伴い，PC上でビデオ編集を行うことが，広く普及してきている． しかし，一方でビデオ編集でネックとなるのが，エンコードにかかる時間である．高解像度，映像の時間が長くなるほど，エンコードにかかる時間は多くなり，作業効率の低下につながる．この問題を解決する手段として近年，一般的になりつつあるデュアルコアCPU，マルチプロセッサを搭載したPC環境で，MPEG2エンコーダをOpenMPにより並列化する． 現在のMPEG2エンコーディングの並列処理は，小高らによるOSCARチップマルチプロセッサ上での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があり，プロセッサ数が４つの場合，逐次実行処理に対して約3.5倍の速度向上が得られている. [9][10]

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，JPEG,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いては，あまり明らかになっていない．MPEG2エンコーディングにおいてGOP，スライス，マクロブロック単位の処理については独立に処理ができるので，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エンコーダを対象としてハイブリッド並列プログラミング による高速化の有効性を実験的に検証することを目的として，OpenMPにより共有メモリ型計算機環境での並列プログラミングを担当し，共有メモリ環境での実験を行った．OpenMPで並列化する処理は，処理量の多いDCT，IDCT，量子化，逆量子化である．また，共同研究者である池上は，ハイブリッド並列プログラミングによる実験を行った．各ノードへのデータ分割をMPIで行い，各ノード内では私がOpenMPにより並列化したプログラムにより８×８画素ブロックを均等に分割して２プロセッサで処理を行う．

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であるmpeg2enocde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て，Main ProfileにおけるLow Level, Main LevelとHigh 1440 Levelの３種類のデータサイズで実験を行った．OpenMPによる並列化アルゴリズムはブロック分割により行い，MPEG2エンコーダのDCT,IDCT,量子化,逆量子化計算を並列化した．ハイブリッド並列プログラミングの実験環境には，１ノードXeon2.8GHz×2,メモリ2GBが16ノードのSMPクラスタ「Atlantis」を使用，OpenMPによる並列化の実験では，Xeon2.8GHz×2,メモリ4GBのマルチプロセッサ環境「Tyrano」で行った．

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エンコーダについて述べ，４章ではOpenMPによるMPEG2エンコーダの並列化の方法について説明し，５章ではOpenMP並列プログラミングによる実行結果を示し，またハイブリッド並列プログラミングでの実行結果と比べて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

一般的な PC クラスタには，図１に示すような共有メモリ型並列計算機，図２に示すような分散メモリ型並列計算機がある．

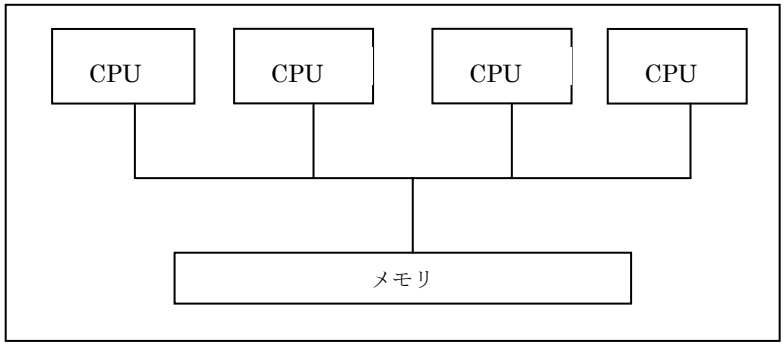

図１:共有メモリ型並列計算機

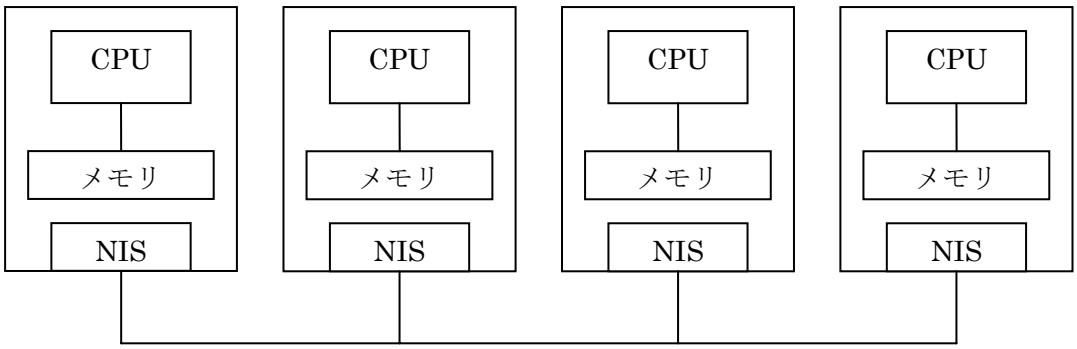

図 2:分散メモリ型並列計算機

PC クラスタにおいて，性能面でのボトルネックとなるのがネットワークといわれている．プロセッサ数が同じであれば，共有メモリ型の方が性能面では優れている．図３に示す SMP クラスタでは，分散メモリ型のように容易にノード数を増やすことができ，ネットワークトラフィックを減らせるために，より高性能化が期待できる．

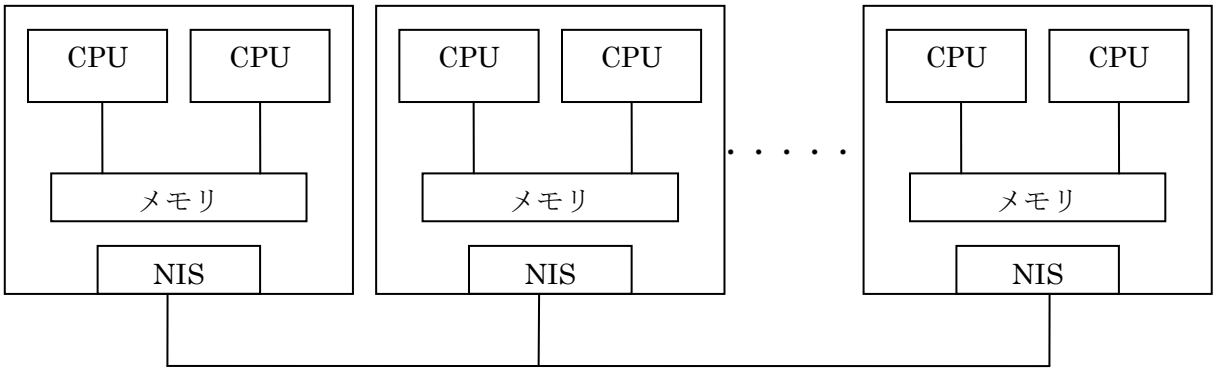

図 3:SMP クラスタ

SMP クラスタは, 分散メモリ環境と共有メモリ環境が混在したクラスタである. 各ノードは共有メモリ環境になっているため, OpenMP により並列化してプログラムの速度向上を図る.

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の形で実現するライブラリレベルでの並列化実装である. このため, C,Fortrun などのプログラミング言語を問わず使用が可能である. 共有メモリ環境での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が多く, ソースコードの互換性もバイナリの互換性もない. 最近になって POSIX thread として多くの UNIX 系 OS に搭載されているが, Windows や他の OS でサポートされていないのが現状である. 一方 OpenMP は, thread と比較して簡単にプログラミングが行え, C/C++,Fortran などのライブラリとして実装され, ベンダー製コンパイラに採用されているため, 標準になっている.

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クラスタ環境において, 分散メモリプログラミングと共有メモリプログラミングの両方の長所生かしたプログラミングである. MPI はノード間で通信を行う並列化実装であるため, ノード内で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. 加えて, OpenMP はノード内でプロセッサが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, ノード間でメモリ共有をする場合には, ソフトウェアでメモリの同期を取らなければならなくなるため, ボトルネックとなる. またすべてのメモリを 1 つの共有メモリ空間として扱うため, メモリの有効活用ができない. したがって SMP クラスタ上において, ノード間では MPI による並列化, ノード内では OpenMP によって並列化する, ハイブリッド並列プログラミングが理想的である. 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 の処理担当範囲を示す.

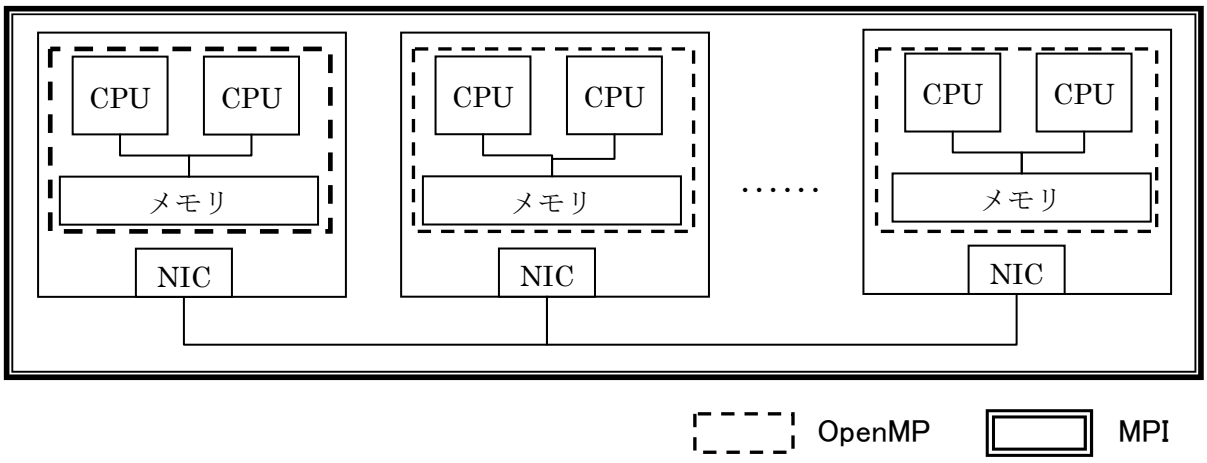

図 4:MPI と OpenMP の処理担当範囲

図 4 において, MPI での並列化ではデータをノード毎に分割して送信する. データの分割の方法として, データを均等に分割するブロック分割, データを細かく分割するサイクリック分割がある. 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示す.

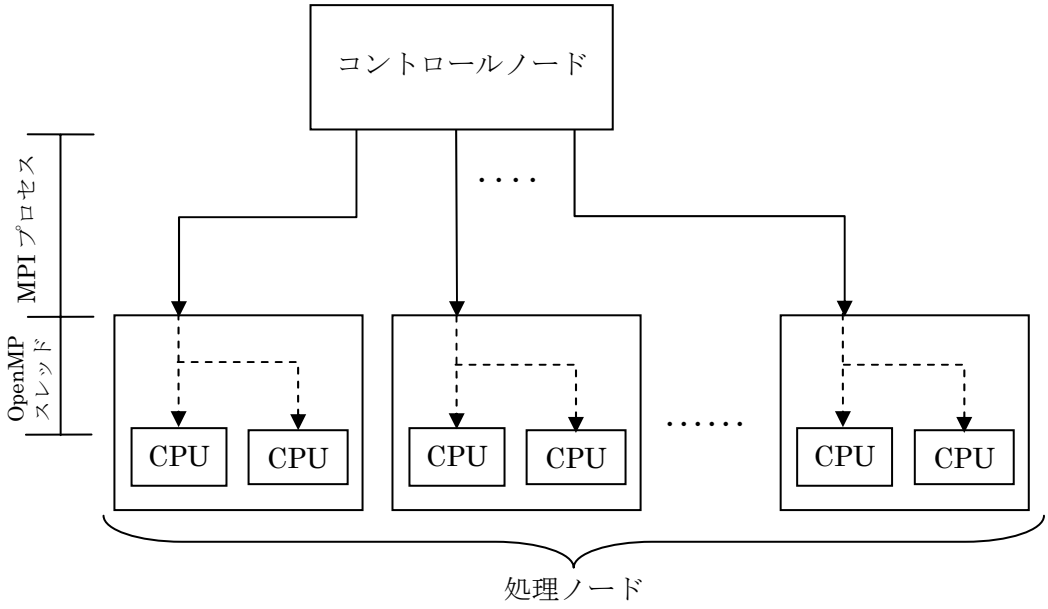

図 5:ハイブリッド並列プログラミング実行イメージ

以上のことから価格・性能比に優れた SMP クラスタとハイブリッド並列プログラミングの組み合わせは非常に有効な手段である.

2.3 OpenMP

OpenMP とは，共有メモリのマルチプロセッサ環境における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る．ベースとなるプログラミング言語（Fortran,C/C++）に指示文（ディレクティブ）を追加することで，並列プログラミングを行う．

OpenMP は，以下のような特徴がある．

・指示文，ライブラリ，環境変数によって，ベース言語を拡張する

・並列化実行部をプログラマが指示

・指示文を無視することで，逐次実行が可能

OpenMP の実行モデルを図 6 に示す．

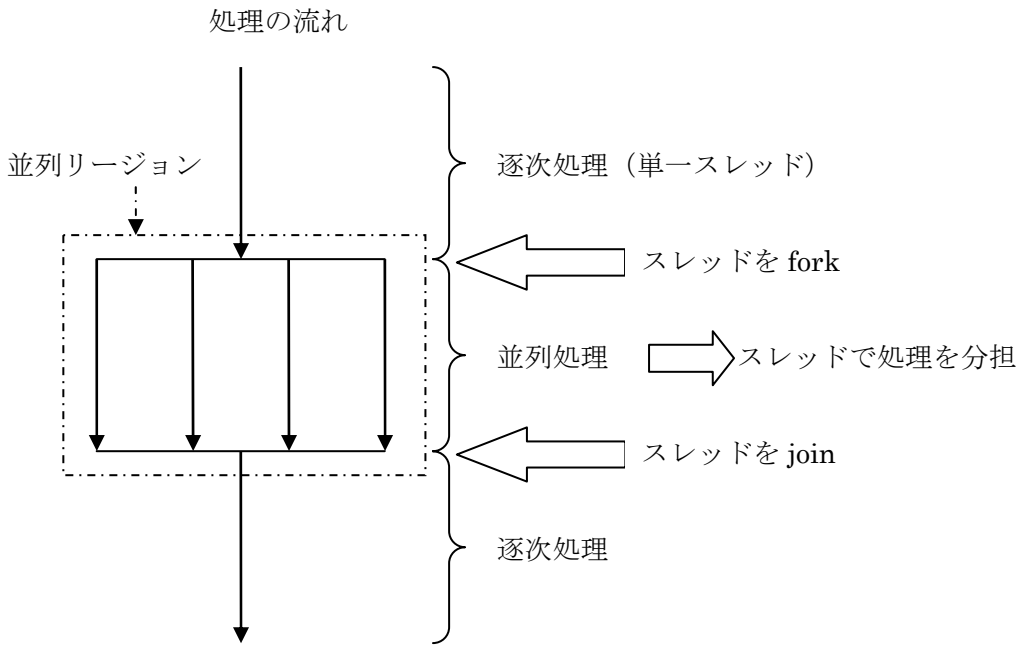

図 6:OpenMP の実行モデル

fork-join モデルとは，複数のスレッド（プロセッサ）が並列プログラムを実行するとき，逐次部分は，単一のスレッド（マスタースレッド）で実行して，プログラムが並列指示文によって並列化された部分になると，ほかのスレッドを生成（fork）して並列処理を行う．並列指示文が終わると，単一スレッドのみの逐次実行（join）に戻る．プログラム中にあるノードの数により fork-join を繰り返す．

OpenMP では，逐次処理部を逐次リージョン（Sequential region），並列処理部を並列リージョン（Parallel region）と呼ぶ．

次に，ベースとなる言語毎の OpenMP の指示文フォーマットを表 1 に示す.

表 1:OpenMP 指示文フォーマット（Fortran,C/C++）

Fortran	**!$OMP** *directive_name*[*clause,clause,*…]
C/C++	**#pragma omp** *directive_name*[*clause,clause,*…]

directive_name:**指示文**,　clause:**指示節**

C/C++言語の場合，1 つのディレクティブには，1 つの *directive_name*（指示文）のみ指定できる．ディレクティブは大文字，小文字を区別する．*clause* の順番に制限はなく，その他の規則は C/C++に従う.

3 MPEG2 エンコーダ

3.1　MPEG2 技術

MPEG2 は，MPEG1 に続くものとして５～１０Mbps 程度のビットレートで現行テレビ放送の品質を実現する符号化方式として，MPEG（Moving Picture Experts Group）により標準化された．MPEG2 では，符号化機能を「Profile」・解像度を「Level」と階層的に分類しており，用途によってそれらを組み合わせて圧縮する．プロファイルは，7 種類存在するが，標準的な利用では MP(メインプロファイル：Main Profile）が用いられる．そのほか SP（シンプルプロファイル：simple profile）などがある．表２に Main Profile における各 Level のパラメータの最大値を示す．

表 2:Main Profile の各 Level 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のデータ構造は，GOP(Group Of Picture)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ち，各フレームはスライス，マクロブロック，ブロックという構造を持っている．

3.2 MPEG2 エンコーダアルゴリズム

図 7 に MPEG2 エンコーダのアルゴリズムを示す．

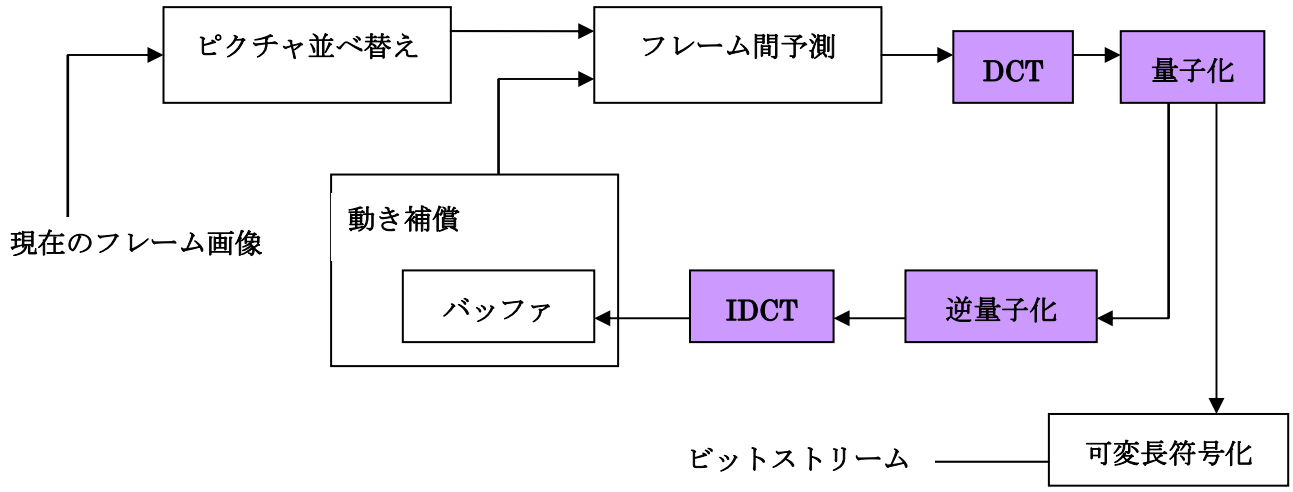

図 7:MPEG2 エンコーダのアルゴリズム

処理ごとの概要は以下の通りである.

(1) フレーム間予測

高い圧縮効率を得るために, 異なる時刻のフレームに基づいて予測画像を生成し, 入力画像と予測画像の差分(誤差)画像を符号化する処理. 一般に, 前方向予測のみを用いて符号化されるフレームがPフレームと呼ばれ, 前方向予測, 後方向予測, 両方向予測のうちいずれかを選択して符号化されるフレームが B フレームである. なお, フレーム間予測を用いずに符号化されるフレームがIフレームである. 図8にフレーム間予測のピクチャタイプの関係を示す.

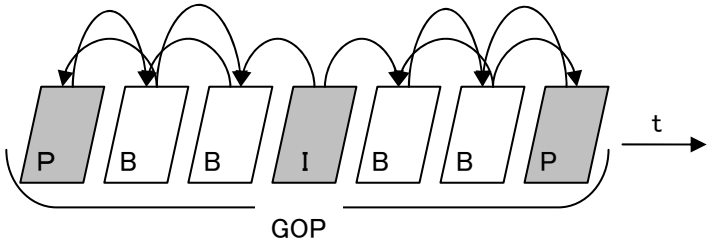

図 8:フレーム間予測とピクチャタイプ

(2)動き補償

フレーム間予測における動きを補うものである.現在のフレームを予測する場合に, 動きの分だけずらした位置の画像を用いるものである.動き補償を行うためには, 画像の動き量を推定する動きベクトル探索が必要になる.符号化する場合には, この動きベクトルも同時に符号化する.

(3)DCT

フレーム間動き補償予測誤差のマクロブロックの6つのブロックについて, 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の削減に用いられる.

(4)IDCT

DCT の逆変換, 圧縮した画像を現画像に近い形で復元する.

(5)量子化

DCT 係数を対応する量子化マトリクスにより除算.

(6)逆量子化

量子化マトリクスによる乗算を行う.

3.3 DCT/IDCT・量子化/逆量子化

並列化を行う MPEG2 エンコーダの処理について次に述べる.

(1)DCT と IDCT

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,

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

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

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

N×N の 2 次元 DCT と IDCT の定義式については次の通りである.

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

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

2 次元 DCT の処理は 1 次元 DCT の処理に次元分解でき, 計算量を減らすことができる. これは, ブロック内の各行を x 方向に 1 次元 DCT を行い, 結果を同じ行に戻し, 次に 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 の定義式は次の通りである.

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

$$X_{k,j} = \sqrt{\frac{2}{N}} C(l) \sum_{k=0}^{N-1} g_{k,j} \cos\frac{\pi l(2j+1)}{2N}$$

定義通りの 8× 8 ブロックの 2 次元 DCT では, 64×64 回の積和計算が必要となるが, 次元分解することで, 16 回の 1 次元 DCT（64×16 の積和）と 1/4 になる.

(2)量子化・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. 8×８の DCT 係数の各場所に異なる感度を与える重みが入って，量子化の粗さを変化させる.デフォルトの量子化マトリクスを図 9 に示す.

（ｉ）イントラ用量子化マトリクス

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	69	83

（ⅱ）非イントラ用量子化マトリクス

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

図 9:デフォルトの量子化マトリクス

4 OpenMP による MPEG2 エンコーダの並列化

4.1 並列化アルゴリズム

(1)DCT, IDCT の並列化アルゴリズム

DCT と IDCT 計算では，１つの画素を変換するためには８×８ブロック内のすべての値が必要である．よってブロックを分割することができず，ブロック毎に分割することもできない．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一のメモリを共有して，ブロックすべてのデータを参照できるため，ブロック分割により８×８ブロック単位をプロセッサごとに割り当てることができ，余分なデータ転送などは起きない．

(2)量子化と逆量子化の並列化アルゴリズム

量子化と逆量子化では，処理する画素毎に独立して計算が行える．よって８×８ブロックをブロック分割により，DCT・IDCT と同様にして並列化を行った．DCT，IDCT,量子化，逆量子化計算の並列化方法を図 10 に示す．

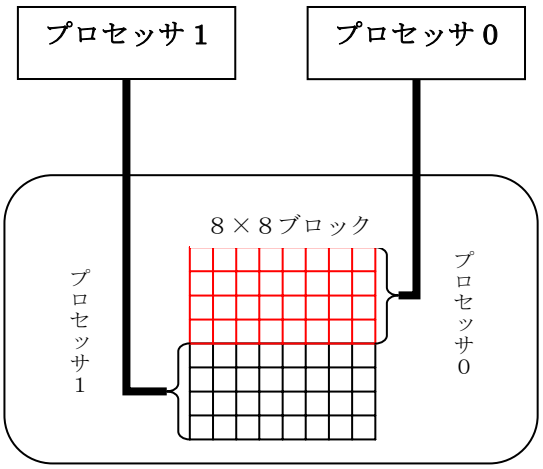

図 10:共有メモリ環境での並列化

4.2 OpenMP による並列化手法

SMP クラスタ上での OpenMP 処理は図 11 に示す通りで，ノード内の共有メモリ環境での実行を OpenMP で並列化する．

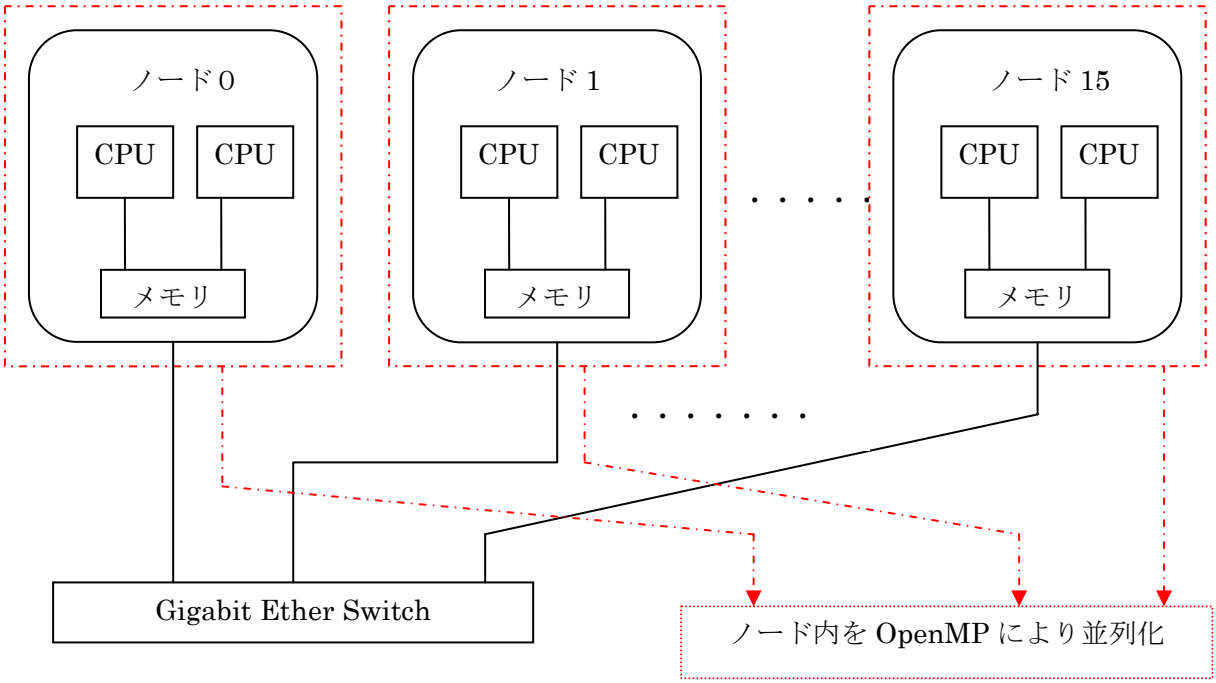

図 11: SMP クラスタ上での OpenMP

また，図 12 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．

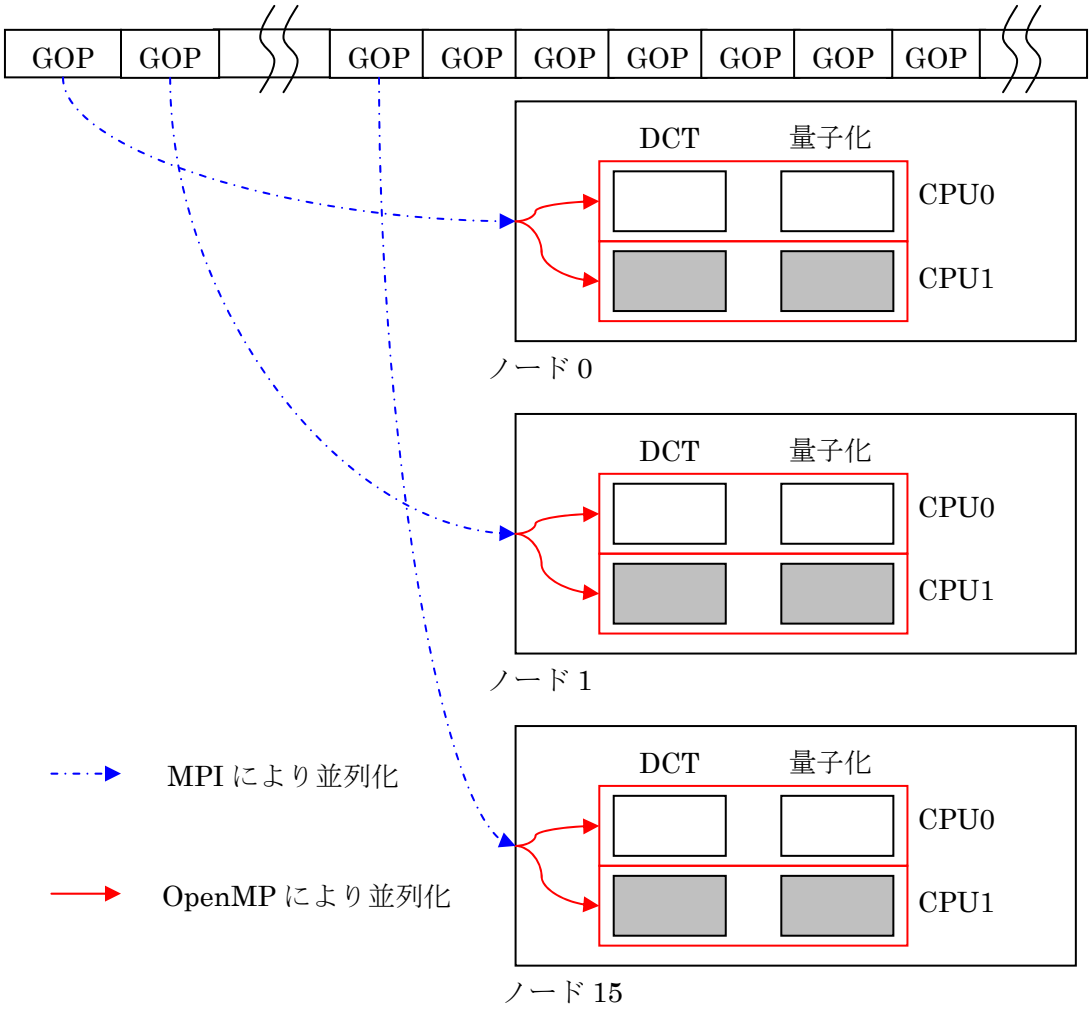

図 12: ハイブリッド並列プログラミングによる並列化

GOP 単位の MPEG2 動画を MPI により図 12 のようにノード毎に分割する．ノード内では GOP の 1 フレームずつの処理を OpenMP によって並列化する．DCT/IDCT・量子化/逆量子化についてはノード毎にスレッドに分割し，それ以外の処理については 1 プロセッサで行う．MPI で GOP 単位を MPI で並列化する方法として，ブロック分割，サイクリック分割がある．ブロック分割によるアルゴリズムを図 13 に示す．

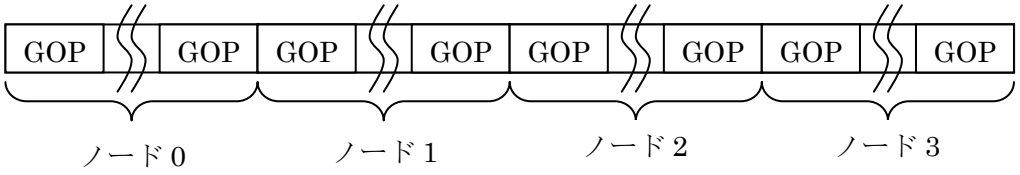

図 13: ブロック分割

図 13 はノード数４の場合である．ブロック分割では，入力動画をノードの台数で分割して並列化する．ブロック分割は，サーバとノードとの通信回数が１度で済むので，並列化において全ての処理範囲が同じ処理量であれば，効率が最も良い．
次に，サイクリック分割によるアルゴリズムを図 14 に示す．

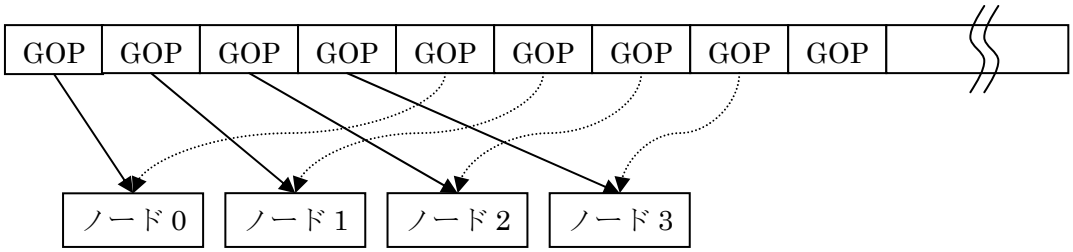

図 14:サイクリック分割

サイクリック分割は，入力動画を小さな単位に区切り，順番にノードに割り当てて並列化を行う．サイクリック分割は，サーバとノードとの通信回数が多くなってしまうが，処理量を分散させることができ，処理範囲毎に計算量が異なる場合に有効である．動画像の場合には，異なる画像の入力がほとんどなので，GOP 毎に処理量が変化する．そのため MPEG2 エンコーダの並列化では，ブロック分割よりもサイクリック分割のほうが効率が良いと考えられる．

5 性能評価と考察

5.1 マルチプロセッサ環境での性能評価

(1)実験環境

実験環境には, Intel Xeon2.8GHz を２台搭載し，４GB のメモリを搭載したマルチプロセッサ環境「Tyrano」で行った．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.

(2)実験条件

使用する Profile は Main Profile で, High 1440 Level, Main Level, Low Level の３種類の実験を行った. Main Level における入力動画は，図 15 のような 720×480 画素の自然動画で, Low Level, High 1440 Level の入力画像は，図 15 の画像を 352×288, 1440×1080 画素にリサンプリング を行った動画を用いた. 動画の長さは 64 秒間，1920 フレームである．さらに各 Level において，同じ入力動画のフレーム数を変化させた場合についても実験を行った.

図 15:入力動画

(3)実行結果

High1440 Level, Main Level, Low Level における，スレッド数１および２についての実行結果を表３に示す.

表３:スレッド数の違いによる Level ごとの実行時間（秒）

スレッド数	1	2
High1440 Level	8965	4735
Main Level	1934	1005
Low Level	630	333

Level 毎の速度向上比を表４に示す.

表４:Level ごとの速度向上

	High 1440　Level	Main Level	Low Level
速度向上	1.89	1.93	1.89

表４からも分かる通り，各 Level において並列化前の段階と比べて，スレッド数に近い速度向上が得られた．次に各 Level でフレーム数を変化させた場合の実験結果を示す．フレーム数は 30flame(1 秒), 300flame(10 秒), 900flame(30 秒)の３通りに変化させて行った．表５に並列化前の実行結果を示し，表６に並列化後の実行結果を示す.

表 5:並列化前（スレッド数１）の実行時間（単位：秒）

並列化前	High1440	Main	Low
30flame	129.1	28.3	7.5
300flame	1432.3	290.7	77.3
900flame	4298.1	903.6	235.1

表 6:並列化後（スレッド数 2）の実行時間（単位:秒）

	High1440	Main	Low
30flame	74.5	16.4	4.4
300flame	751.8	157.7	45.4
900flame	2231.4	480.1	136.2

各 Level のフレーム数の違いによる速度向上を表 7 に示す.

表 7:フレーム数の違いによる速度向上

速度向上	High1440	Main	Low
30flame	1.73	1.72	1.69
300flame	1.91	1.84	1.70
900flame	1.93	1.88	1.73

各 Level においてフレーム数の違いによる速度向上の結果が得られた．Low Level では，他の Level と比べると速度向上の割合が小さい.

5.2 ハイブリッド並列プログラミングでの性能評価

(1)実行環境

実行環境には, Intel Xeon2.8GHz を 2 台搭載した SMP ノード 16 台を Gigabit Ether で接続した. 合計 32 プロセッサの SMP クラスタ「Atlantis」を用いる. Atlantis の構成図を図 16 に示す.

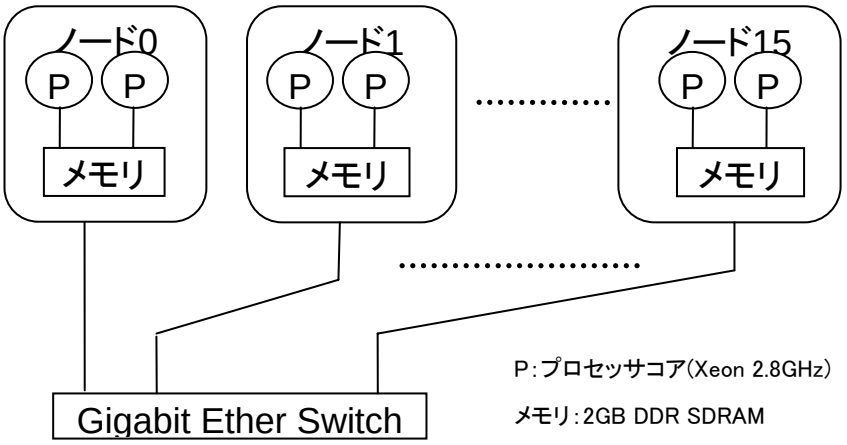

図 16:Atlantis の構成図

(2)実験条件

実験条件は, マルチプロセッサ環境と同じで, 3 種類の Level それぞれ 1920 フレームの入力動画である.

(3)実行結果

実行結果を表 8 に示す.表中の MP@LL は Main Profile の Low Level を示し, MP@ML は Main Level を示し, MP@HL は High 1440 Level を示す. また Low Level での実行時間のグラフを図 17 に, Main Level の実行時間を図 18, High 1440 Level の実行時間を図 19 に示す. 1 ノード当たりのプロセッサコア数は 2 で, 16 ノードでは 32 プロセッサコアを使用した実行結果である.[5]

表 8:実行時間（秒）

	ノード数	1	2	4	8	16
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	183.1	97.5	63.4	63.7
MP@ML	ブロック分割(ハイブリッド)	1248.5	629.3	322.5	184.9	158
	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	247.7	154.3
MP@HL	ブロック分割(ハイブリッド)	6014.4	3094.3	1543	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

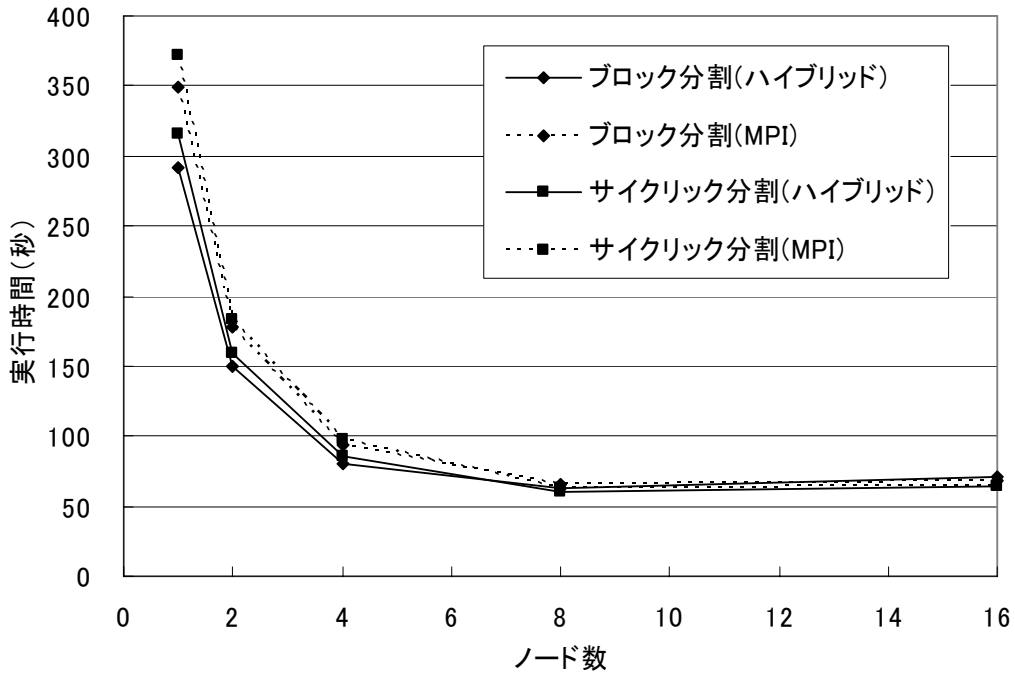

図 17:Main Profile, Low Level の実行結果

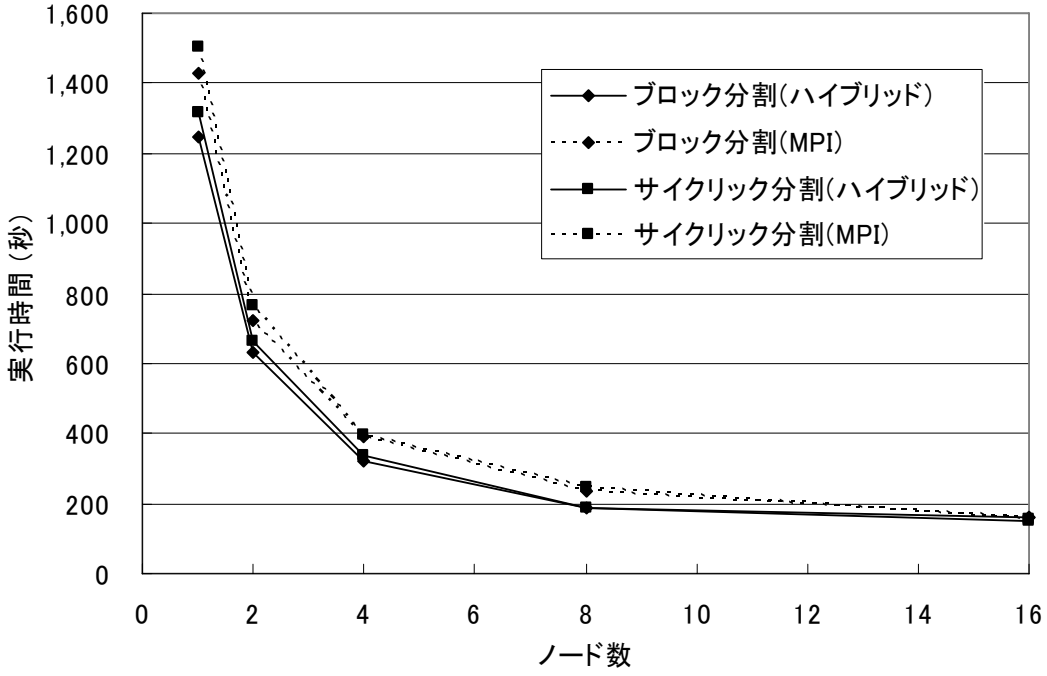

図 18:Main Profile, Main Level の実行結果

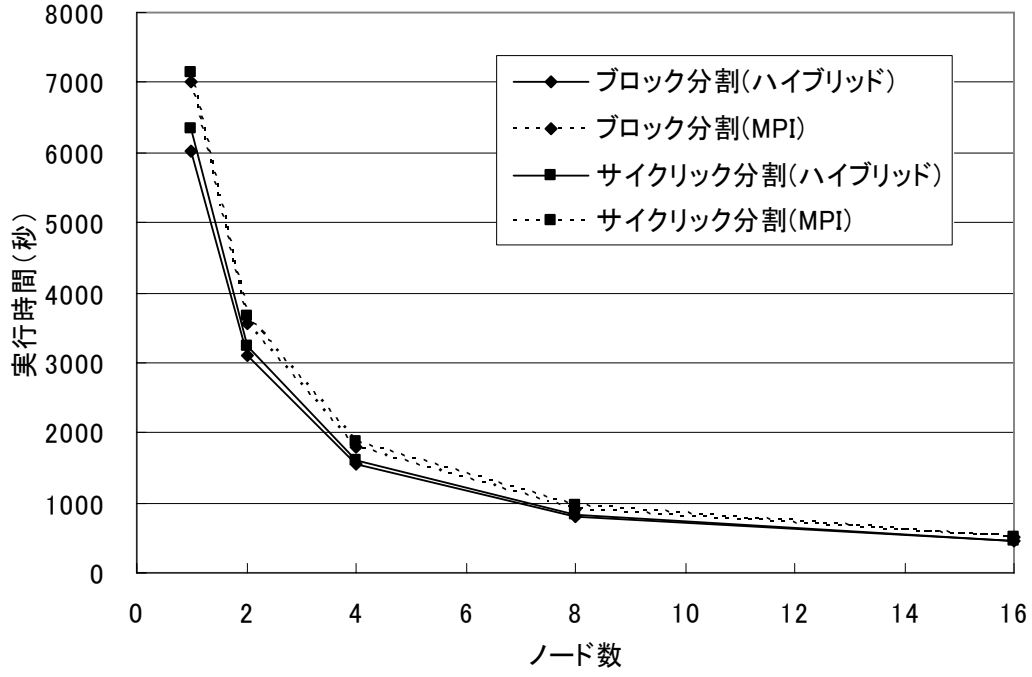

図 19:Main Profile, High 1440 Level の実行結果

16 ノードでの Low Level の結果を除き，ハイブリッド並列プログラミングの結果の方が MPI のみの場合に比べて常に実行時間が短くなっていることが分かる． 特に 1 ノードでの実行時間は，ハイブリッドにより大幅に短縮されている．またブロック分割とサイクリック分割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くなっているが，16 ノードではサイクリック分割の方が短くなっている．

次に，Low Level の速度向上を図 20，Main Level での速度向上を図 21，High 1440 Level での速度向上を図 22 に示す．

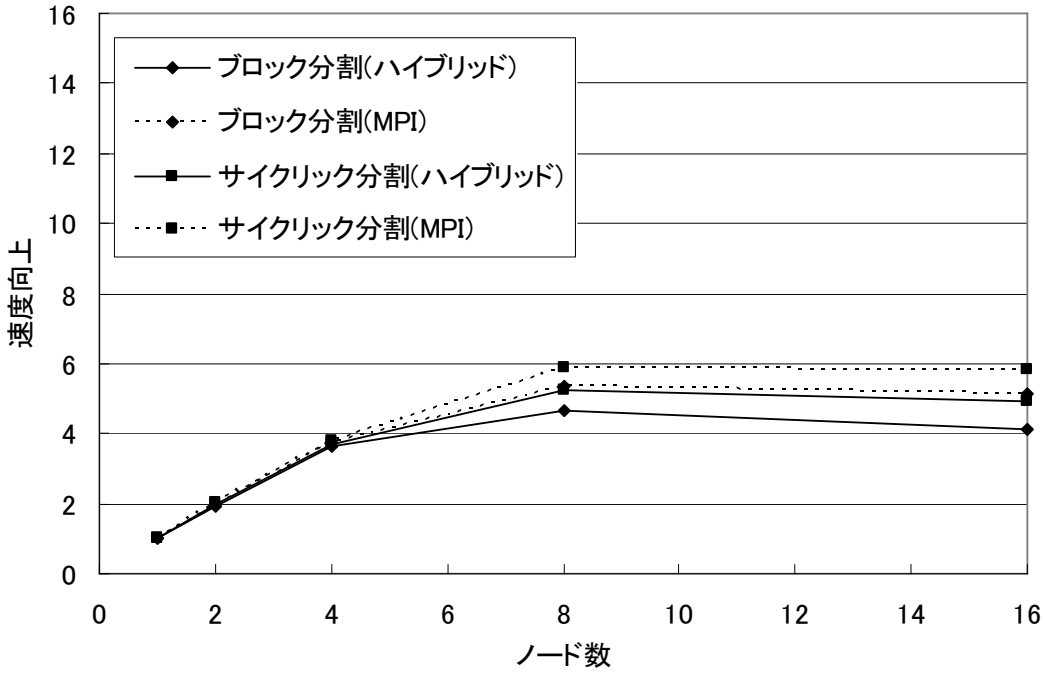

図 20:Main Profile, Low Level での速度向上

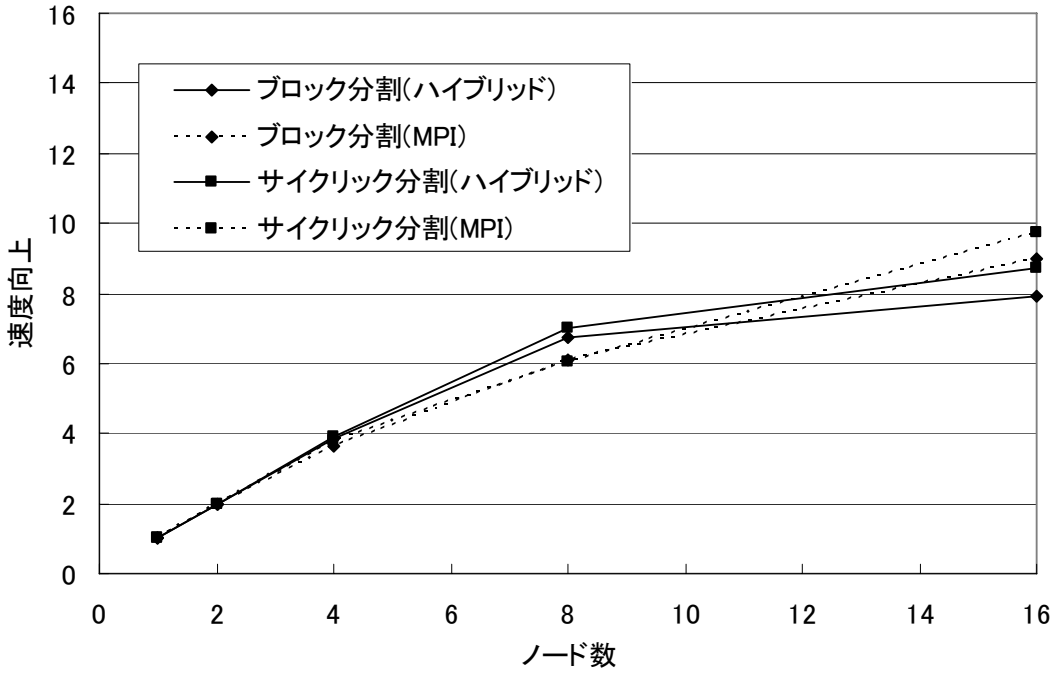

図 21:Main Profile, Main Level での速度向上

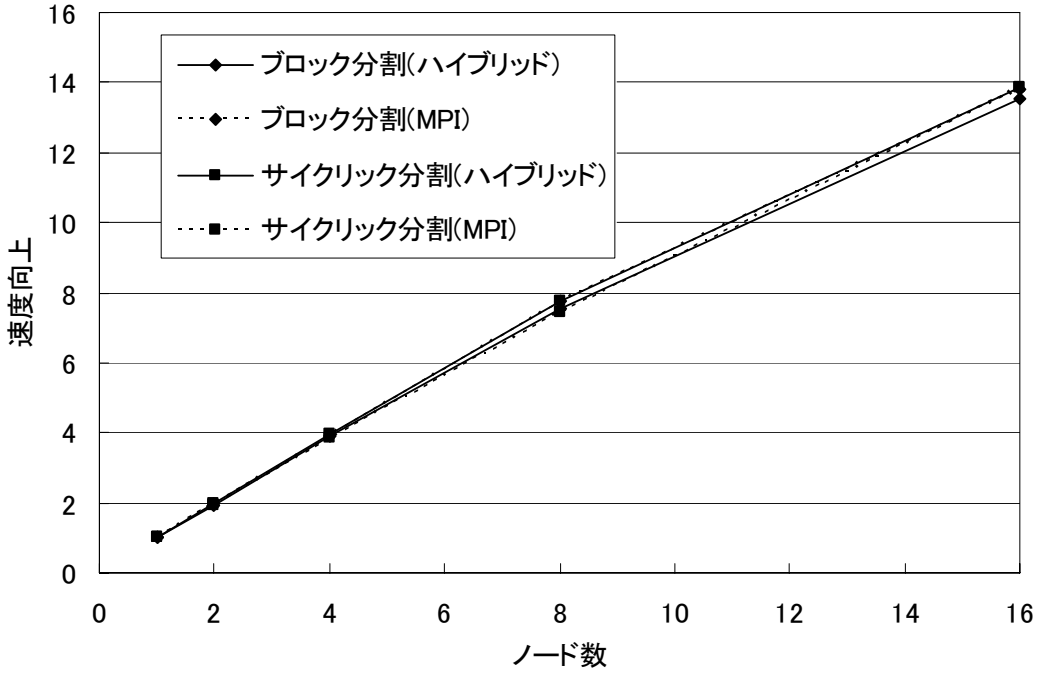

図 22:Main Profile, High 1440 Level での速度向上

速度向上はブロック分割による結果の方が高く，Low Level では 8 ノードで約 5.8 倍，Main Level では 16 ノードで約 9.7 倍，High 1440 Level では 16 ノードで約 13.9 倍になった．

5.3 考察

OpenMP による MPEG2 エンコーダの並列化では，スレッド数 2 で実行した場合において約 2 倍近い速度向上が得られた．表 7 において，Low Level での速度向上の数値が他の Level に比べて少ないが，原因としては，Low Level では画像サイズが小さいため，DCT・量子化計算が他と比べて少なく，並列化効果が少ないと考えられる．計算量が増えれば速度向上が上がるのは，表 7 の同一 Level でのフレーム数が増える場合において，比率がわずかだが上昇していることで推察できる．しかし表 4 の結果より，フレーム数が増えるにしたがって常に速度向上が得られるわけではなく，1920 フレーム数の結果より最終的には 1.9 倍ほどの速度向上に留まることが分かる．

また，ハイブリッド並列プログラミングの速度向上図 20-22 において，Low Level での 16 ノード実行で速度向上が見られなかった．これはマルチプロセッサ環境でフレーム数が少ない場合において，速度向上の割合が他の Level に比べて小さいことと深く関係していると思われる．並列化による通信回数の増大が，計算処理の並列化効果以上に影響していると考えられる．しかし High 1440 Level においては，ノード数の増加と共に速度向上が見られており，ハイビジョン動画像の処理では，ハイブリッド並列プログラミングの効果は大きいと考えられる．

以上のことから，ハイブリッド並列プログラミングの効果を得るためには，ある程度のデータ量と，フレーム数が必要となる．High 1440 Level において理想的な速度向上が得られている．これは，1 フレーム当たりのデータ量が多く計算量が大きくなるためであると推察できる．

今後の課題として，High 1440 Level におけるハイブリッド並列プログラミングの更なる有効性の検討のために,様々な動画像を用いた実験が必要であると考える．例えばスポーツなどの動きの激しい動画像と，動きの少ない動画像の比較,またアニメーションなどのデジタルカメラ以外の動画像を用いた実験である．本研究では同一動画像を用いて，Main Profile の 3 つの Level における実験を行った．これから普及していくであろう High 1440 Level に重点を置くことで，MPEG2 エンコーダにおけるハイブリッド並列プログラミングの有効性を高めることができる．

6.おわりに

本研究では，SMP クラスタ上での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に関連して，共有メモリ環境での OpenMP による並列化を行った．Main Profile の 3 つの Level についてマルチ CPU 環境で実行時間を計測して，約 1.9 倍の理想的な速度向上が得られた．

また，ハイブリッド並列プログラミングにおいて最大の速度向上として，Low Level で 5.8 倍(8 ノード), Main Level で 9.7 倍(16 ノード), High 1440 Level では 13.9 倍(16 ノード)の速度向上が得られた．計算処理が少ない(データサイズが小さい)場合では，ハイブリッド並列プログラミングによる効果が小さいことが分かった．

今後の研究課題として 更なるエンコーダの高速化が挙げられる．特に High1440Level でのエンコードでは，今回並列化した場合でも実行時間が実際の映像時間と比べて約７５倍のエンコード時間を要する．これでは余りに非効率的である．今回実験に用いたプログラムは，MPEG2 エンコーダのリファレンス実装である．このプログラムは cos 計算に，math.h を用いてマクローリン展開を使用した関数を使っており，エンコーダ自体には高速化アルゴリズムは採用されていない．高速化アルゴリズムを用いることで，更なる実行時間の短縮が期待できる．

現在普及しつつあるデュアルコアの CPU では，並列処理は必要不可欠なものとなる．高精細な動画についての処理の高速化と並列処理を組み合わせれば，快適な動画編集環境を構築することができる．

謝辞

先ず，本研究の共同研究者である池上氏に心より感謝致します．そして，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，本研究にあたり，貴重な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．

さらに，メーリングリストでの質問に関して，貴重な助言を頂ました筑波大学の佐藤三久教授に心より感謝致します．

参考文献

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

[2]OpenMP: http://www.openmp.org

[3]Omni OpenMP Compiler: http://www.hpcc.jp/Omni/home.ja.html

[4]日本 SGI:“OpenMP Directives for C”,
http://www.issp.u-tokyo.ac.jp/super/manual/system-b/29_OpenMP_C_J.pdf

[5]池上広済:“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”,立命館大学理工学研究科修士論文,2006

[6]柿下裕彰:“PC クラスタ上での OpenMP 並列プログラミング[Ⅰ]”,立命館大学理工学部情報学科卒業論文, 2000

[7]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

[8]黒川耕平:“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

[9]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.

[10]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

[11]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

[12]佐藤三久:“OpenMP,JSPP'99 OpneMP チュートリアル資料”,1999

[13]半谷精一郎,杉山賢二:“JPEG・MPEG 完全理解”,コロナ社, 2005

[14]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

[15]MPEG: http://www.mpeg.org

[16]三木光範　他:“PC クラスタ超入門 2000”,PC クラスタ型並列計算機の構築と利用,超並列計算研究会,2000,http://mikikab.doshisha.ac.jp/dia/smpp/cluster2000