卒業論文

PC クラスタ上での OpenMP による JPEG2000 エンコーダの並列化

氏　　名：　宮城　雅人
学籍番号：　2210990210-5
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

本研究では、PC クラスタ上の OpenMP により、JPEG2000 エンコーダの並列化を行う。利用する PC クラスタシステムとしては、SCore 用いる。OpenMP は共有メモリ向けのプログラミングモデルであるが、SCore 上のソフトウェア分散共有メモリ(SCASH)と、SCASH に対応している Omni OpenMP コンパイラを利用することで、PC クラスタ上での OpenMP プログラミングが可能である。

JPEG2000 ではリファレンスソフトウェアとして、JasPer が公開されている。JasPer は C 言語で実装されており、本研究では、この JasPer を SCore の Omni-SCASH 環境へ移植し、OpenMP による並列化を行った。その過程で、分散共有メモリの動的確保と解放を行う、omscmmm_malloc(), omscmmm_free()の設計と実装を行った。これにより、共有メモリ環境とほぼ同一の OpenMP プログラミング環境を手に入れることができた。

JPEG2000 では、画像を複数のタイルと呼ばれる領域に分割する。OpenMP による並列化は、タイルを PC クラスタの各ノードのプロセッサ(スレッド)に分担してエンコードさせるという方法で行った。

そして、実際に JPEG2000 のエンコードを行い、並列化の効果を検証した。エンコードは、タイルのサイズと OpenMP による繰り返しの各スレッドへの割り当て方法を、それぞれ変化させて行った。その結果、スレッド数 1 に対して、スレッド数 16 で最大約 11 倍の速度向上を得ることができた。本論文の最後にこの結果を考察し、ソフトウェア分散共有メモリ上での OpenMP プログラミングであることをふまえた、更なる高速化方法と、タイル分割とは別の JPEG2000 の並列化方法について述べる。

目次

図目次

表目次

1. はじめに

近年、汎用の PC と汎用ネットワークを用いた計算機クラスタ（PC クラスタ）が、コスト面だけでなく、性能の面でも大きな利点をもつようになってきた。

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たシステムが複数個接続され、プロセッサは他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モデル。プロセッサは他のプロセッサの主記憶を読み書きできる分散共有メモリモデルである。分散メモリモデルの並列計算機における並列プログラミングライブラリとしては、MPI(Message Passing Interface)や PVM(Parallel Virtual Machine)が広く用いられている。また、共有メモリモデルの並列計算機においては、OpenMP[4]が主流になってきている。

PC クラスタの中に、新情報処理開発機構の RWC(Real World Computing)プロジェクト[2]で開発され、現在 PC Cluster Consortium[1]で開発が続けられている SCore クラスタがある。SCore は PC で動作する OS である Linux 上に構築された並列計算環境であり、ゼロコピー通信を用いた高速・低遅延通信ライブラリである PM II 通信ライブラリをもつ。さらに、ソフトウェア分散共有メモリシステム（SCASH）をもち、分散メモリモデルである PC クラスタ上で共有メモリプログラミングを可能にする、分散共有メモリモデルとして利用できる。SCASH と Omni OpenMP コンパイラ[5]を用いることにより、PC クラスタ上で OpenMP プログラミングが可能となっている。

また、画像圧縮フォーマットの分野では、従来の JPEG の後継として、JPEG2000 が規格化されている[9,10]。JPEG2000 は従来の JPEG と比較して、高圧縮率、高品質で様々な機能をもつなど、優れた点が多い。しかし、エンコード・デコード時の処理負荷が大きいという側面もある。

本研究では、高解像度の画像の JPEG2000 エンコードを、SCore PC クラスタ上の OpenMP 環境において並列化することにより、高速でストレスのない時間内でのエンコードを可能にすることを目的とする。2 章において SCore PC クラスタの概要と OpenMP の説明、3 章にて JPEG2000 の概要とアルゴリズムの説明を行い、4 章にて SCore PC クラスタ上での JPEG2000 エンコードの並列化の方法を説明する。そして、5 章にて 16 台の SCore PC クラスタにおける並列化の実行結果と効果を示し、結果を考察する。

2. PC クラスタ上での OpenMP

2.1. PC クラスタ

PC クラスタとは、近年安価になった PC をネットワークにより複数台接続し、分散処理・並列処理を行うシステムである。クラスタには高性能計算を目的とした HPC(High Performance Computing)クラスタと高可用性を目的とした HA(High Availability)クラスタがあるが、本研究では前者の HPC クラスタを用いる。

HPC クラスタシステムは通常あるノードから別のノードのメモリに直接アクセスできない、分散メモリモデルとなる。また、各ノード間の通信には高速・低遅延なネットワークと通信ライブラリが必要になる。分散メモリモデルでは、MPI(Message Passing Interface)や PVM(Parallel Virtual Machine)といったプログラミングライブラリが広く用いられている。本研究で用いる OpenMP(2.2.で説明する)は共有メモリモデルを対象としており、OpenMP を利用するには、通常の LAN で用いられている 100Mbps の 100Base-TX や 1Gbps の Gigabit イーサネットよりも、更に高速で低遅延なネットワークが必要となる。

PC クラスタシステムでは、単純に複数の PC をイーサネットで接続し、TCP/IP を用いるという構成でも構築可能だが、高速・低遅延のネットワークやジョブ制御、OpenMP を利用するには、専用のミドルウェアが必要となる。本研究では、新情報処理開発機構の RWC(Real World Computing)プロジェクトにおいて開発され、現在 PC Cluster Consortium において開発が続けられている SCore をミドルウェアとして用い、Myricom 社[3]が開発した Myrinet2000 を高速通信ネットワークとして用いている。

本研究に用いた PC クラスタの構成を図 1 に示す。

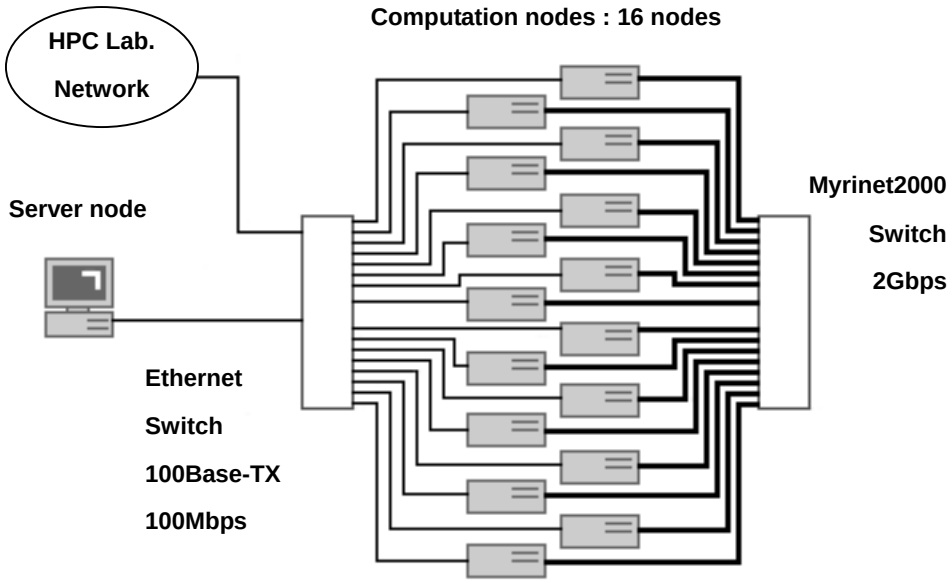


図 1 研究室 PC クラスタの構成

図 1 に示すとおり、SCore を用いた PC クラスタは、サーバーノード(Server node)と計算ノード(Computation nodes)により構成される。サーバーノードはユーザーがログインして PC クラスタの利用を行うノードであり、ジョブの投入や制御を行う。それに対して計算ノードは実際の計算を行う PC ノード群である。

本研究に用いた PC クラスタのサーバーノードの構成を表 1 に、計算ノードの構成を表 2 に示す。

表 1 研究室 PC クラスタ サーバーノードの構成

CPU	Pentium III (Katmai) 450MHz
メインメモリ	512MB
ネットワーク	100Base-TX イーサネット
OS	RedHat Linux 7.3
SCore バージョン	SCore 5.2.0
ノード数	1

表 2 研究室 PC クラスタ 計算ノードの構成

CPU	Pentium III (Katmai) 500MHz
メインメモリ	512MB
ネットワーク	100Base-TX イーサネット, Myrinet2000
OS	RedHat Linux 7.3 + SCore PM モジュール
SCore バージョン	SCore 5.2.0
ノード数	16

全てのノード（サーバー・計算ノード）は 100Base-TX(100Mbps)のイーサネットで接続され、ジョブの投入や制御に用いられる。また、計算ノードはそれぞれ、2Gbps の高速・低遅延ネットワーク Myrinet2000 で接続され、計算中の通信はこのネットワークを用いる。これにより、分散メモリモデルの並列計算システムにおける最大のボトルネックである実行中の通信を高速に行える。

次に、SCore クラスタシステムのソフトウェアアーキテクチャ構成を図 2 に示す。図 2 のように、各ノードには Ethernet と Myrinet の NIC(Network Interface Card)を装備した PC の Linux OS 上に SCore のミドルウェアが構築されている。

以下、本研究において重要な、SCore のコンポーネントについて説明する。

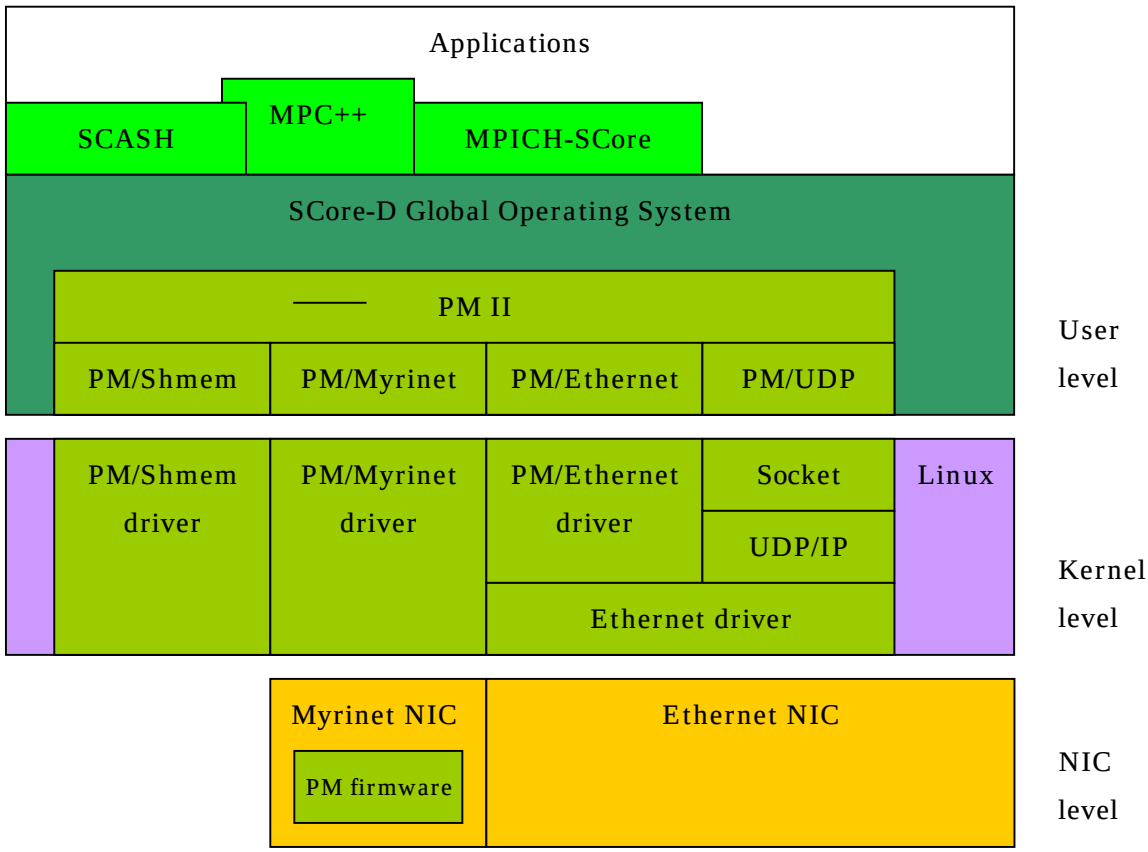


図 2 SCore のソフトウェアアーキテクチャ

- PM II
  クラスタコンピューティング用低レベル通信ライブラリ。Myrinet, Ethernet, UDP/IP, Shmem(SMP での共有メモリ)用のドライバが実装されており、PM II API により、統一的な方法でアクセスできる。ユーザーレベルでのゼロコピー通信（カーネルレベル空間とのメモリコピーをしない）により、高速・低遅延を実現している。
- SCore-D
  プロセッサやネットワークデバイス等の資源管理、ジョブスケジューリング等を行うユーザーレベルのグローバルオペレーティングシステム。ユーザーのジョブを柔軟にスケジューリングし、チェックポイントの機能も持つ。
- SCASH
  PM II を用いたソフトウェア DSM（Distributed Shared Memory, 分散共有メモリ）システム。Omni OpenMP コンパイラを用いることにより、PC クラスタ上での OpenMP プログラムのコンパイル・並列実行を行うことが出来る。

本研究では、SCore 上のソフトウェア分散共有メモリである SCASH と Omni OpenMP コンパイラ(以下 Omni-SCASH 環境と呼ぶ)を用いて、OpenMP による共有メモリモデルでの並列プログラミングを行う。

2.2. OpenMP

OpenMP[4,8]とは、共有メモリモデルのマルチプロセッサにおける並列プログラミングモデルである。ベース言語(Fortran, C/C++)を指示文(directive)とライブラリ関数により拡張し、並列プログラミングを行う。

OpenMP は、従来のメッセージパッシング型のプログラミングモデルである MPI, PVM と比較して、以下のような特徴がある。

- 共有メモリプログラミングモデルなので逐次からの移行が簡単
- 段階的(incremental)に並列化を行うことが可能
- コンパイラが指示文を無視することにより、逐次環境でも同じソースコードを再コンパイルすることで実行が可能(逐次版と並列版を同じソースコードで管理可能)
- ベース言語の拡張であるので、新しい言語ではない（ベース言語を知っていればはじめから新しく学習しなくてよい）

ただし、OpenMP は自動並列化ではなく、プログラマが明示的に並列実行部を指示する。

OpenMP の実行モデルは、図 3 のような fork-join モデルとなっている。

fork-join モデルとは、複数のスレッド(プロセッサ)のチーム(team)が並列プログラムを実行するとき、逐次実行部分はチームの中の一つのスレッド(マスタースレッド)で実行し、並列化指示文により並列に実行する部分になると、他のスレッドを生成(fork)し並列実行を行う。そしてこの並列実行部分が終わるとバリア同期を行い、マスタースレッドのみの逐次実行部分に入る(join)。更にまた並列実行部分があるときは、fork-join を繰り返す。

OpenMP では、これらの逐次実行部分を逐次リージョン(Sequential region)、並列実行部分を並列リージョン(Parallel region)と呼ぶ。

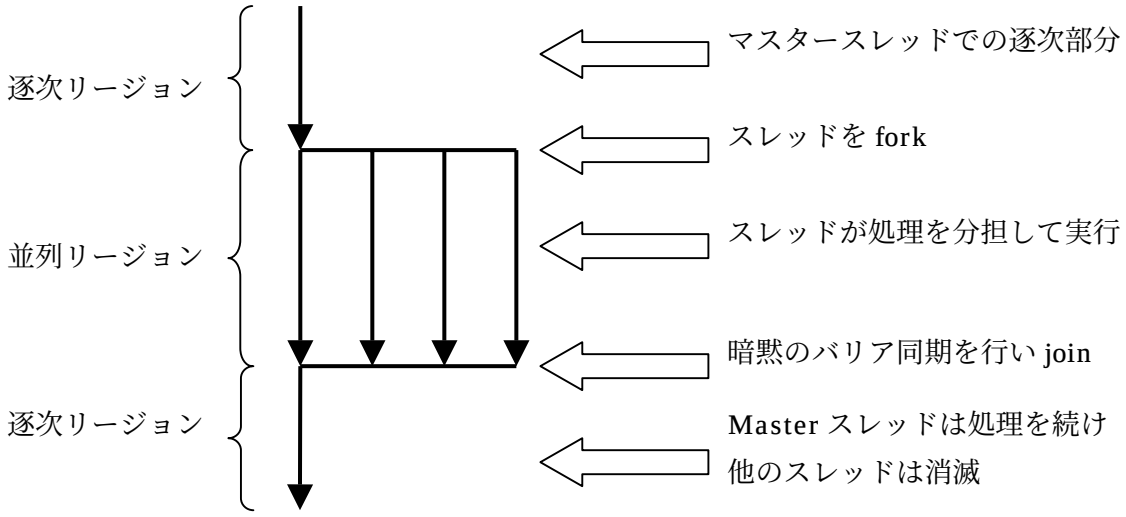


図 3 OpenMP の fork-join モデル

次に、OpenMP の指示文について説明する。表 3 に C/C++と Fortran での OpenMP 指示文のフォーマットを示す。

表 3 OpenMP 指示文のフォーマット(C/C++, Fortran)

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

directive_name : 指示子名, *clause* : 指示節

表 3 の *directive_name* (指示子名)は、並列リージョンを指示する parallel や、スレッド間での同期を行う barrier、マスタースレッドのみで実行する ordered 等がある。また、*clause* (指示節)は、並列ループのスケジューリングを指示する schedule や、変数を共有メモリに置くか、スレッド内に持つかといったことを指示するデータスコープ指示節の shared, private 等がある。

この OpenMP 指示文によって、C/C++言語の場合、並列リージョンは図 4 parallel 指示文による並列リージョン(C/C++言語)のように記述される。

```
#pragma omp parallel
{
    …                    …
    …  Parallel region   …
    …                    …
}
```

図 4 parallel 指示文による並列リージョン(C/C++言語)

OpenMP には様々な指示文があるが、以降では C/C++言語での本研究で用いた指示文等、重要なものについて説明を行う。また Fortran の場合の記述は省略する。

#pragma omp parallel 指示文で指示されたブロック内で、実際の並列化実行を指示する指示文は以下の 2 つである。

- #pragma omp for : for ループの繰り返し(iteration)を分割して並列実行する
- #pragma omp sections : 別々の処理を同時に実行するタスク並列を行う

本研究では#pragma omp parallel 指示文と#pragma omp for 指示文を組み合わせた、#pragma omp parallel for 指示文により並列化を行っている。図 5 にこの記述法を示す。

```
#pragma omp parallel for [clause, clause, ...]
for (init-expr; var logical-op b; incr-expr) {
        …                       …
        …  Parallel region      …
        …                       …
}
```

図 5 parallel for 指示文による並列化の記述(C/C++言語)

for 指示文による並列化が指定できる for ループには制限があり、正規形(canonical shape)と呼ばれる形でなければならない。正規形とは、図 5 における *init-expr,* var *logical-op* b, *incr-expr* に対する制限であり、詳細は省略するが、var は符号付整数変数、b はループ内で不変な整数式、*init-expr* は加減演算子による単調で等幅な増減でなければならない等といったきまりである。さらに、for ループ内では break 文を利用してはならない。Omni-SCASH 環境では return 文でもコンパイルエラーが発生し、利用できない。

次に、for 指示文の指示節(*clause*)において、本研究で利用している schedule 指示節について説明する。schedule 指示節は、for ループの繰り返しをスレッドに割り当てる方法を指定する。指示節の記述は schedule(*kind*[, *chunk_size*])となる。*Kind* に指定できる種類を表 4 に示す。

表 4 schedule 指示節に指定できる種類

static	*chunk_size* で指定されたサイズのチャンクに分割される。各チャンクは、スレッドの番号順にラウンドロビン形式で静的に割り当てられる。*chunk_size* が非指定の場合、繰り返しはほぼ同じサイズのチャンクに等分割され、各スレッドに 1 つのチャンクが割り当てられる。
dynamic	*chunk_size* 回の繰り返しのチャンクを各スレッドに割り当てる。スレッドに割り当てられたチャンクの処理が終了すると、残りのチャンクがなくなるまで、動的に別のチャンクをスレッドに割り当てる。*chunk_size* の既定値は 1。負荷分散に有効である。
guided	繰り返しのチャンクを徐々に小さくしながらスレッドに割り当てる。スレッドに割り当てられたチャンクの処理が終了すると残りのチャンクが無くなるまで動的に別のチャンクをスレッドに割り当てる。チャンクのサイズは指数的に *chunk_size* まで小さくなる。*chunk_size* の既定値は 1。負荷分散に有効である。
runtime	環境変数 OMP_SCHEDULE により、上記何れかの指定を実行時に行う。

3. JPEG2000

3.1. JPEG2000 とは

JPEG2000(Joint Photographic Experts Group 2000)[9,10]とは、西暦 2000 年に規格化された静止画圧縮フォーマットで、従来の JPEG の後継となるものである。JPEG2000 は ISO/IEC 15444 (ITU-T Rec. T.800)で国際規格化されている。

従来の JPEG では、変換アルゴリズムとして DCT（Discrete Cosine Transform, 離散コサイン変換)、符号化アルゴリズムとしてハフマン符号化を用いていたが、JPEG2000 では、それぞれ、DWT（Discrete Wavelet Transform, 離散ウェーブレット変換）と EBCOT(Embedded Block Coding with Optimized Truncation)を用いることが特徴である。

JPEG2000 規格は 2003 年 2 月現在、Part1～Part11 まであり、Part7 は欠番となっている。各 Part のタイトルとその内容を表 5 に示す

表 5 JPEG2000 規格の構成

Part	タイトル	内容
Part 1	Core coding system	画像を JPEG2000 に符号化/復号化するシステム
Part 2	Extensions	拡張部の規定（Part1 の補強）
Part 3	Motion JPEG2000	JPEG2000 の動画規格
Part 4	Conformance	画像伝送におけるコードストリームの検証方法を規定
Part 5	Reference software	参照ソフトウェア(JPEG2000 の機能を検証するための公式ソフトウェア)を決定する
Part 6	Compound image file format	絵と文字が混ざった画像の規定（写真・図・表など目的に応じた画像のファイル形式を規定する）
Part 8	JPSEC (security aspects)	セキュリティを考慮した JPEG2000（作品の不正転載防止による著作権保護やアイドルコラージュ、作者サイン書き換えなどによる改竄防止）
Part 9	JPIP (interactive protocols and API)	双方向性を実現するツール、API、プロトコル（JPEG2000 で融通の利く(例えば ROI の処理)ソフトを如何に作るか、アプリケーションソフトや OS へ如何にして JPEG2000 の機能を組み込むか、ネットワーク上のデータのやり取りはどうするか）
Part 10	JP3D (volumetric imaging)	3 次元画像、浮動小数点データの考慮
Part 11	JPWL (wireless applications)	携帯電話や PHS、プロジェクタ、その他の無線機器などの対応をどうするかの策定

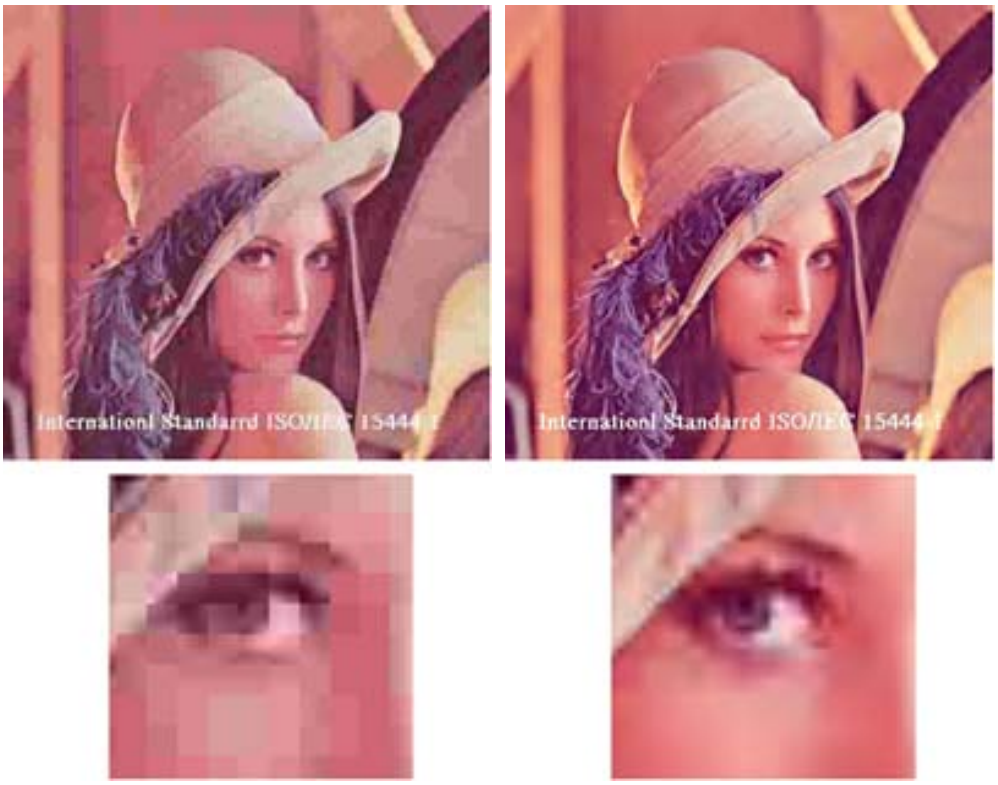
また、従来の JPEG と比較して、JPEG2000 には多数の利点がある。その利点を表 6 に示す。

表 6 従来の JPEG と比較した JPEG2000 の利点

利点	具体的に JPEG2000 でできること
高圧縮	同じ画質なら従来の JPEG よりも圧縮後のファイルサイズを 30%～50%小さくできる。
高品質	かなり圧縮率を高くしても比較的原型をとどめ、見た目の劣化が少ない。
	ブロックノイズ、モスキートノイズが生じず、画像がなめらかになる。
高機能	アニメーションが正式規格になっている(Part3)。
	プログレッシブ表示が正式規格になっている。
	台詞や説明文、撮影日時などを好きに表示および除去できる(Part6)。
広自由度	可逆圧縮(lossless)/不可逆圧縮(lossy)をユーザーが自由に指定でき、可逆圧縮では圧縮率、圧縮後のファイルサイズを自由に指定できる。
	ひとつのファイルからユーザーの回線速度や利用環境に応じて、小さな画像大きな画像まで、様々な解像度で送信が可能。
低ストレス	ROI により、画像の重要な領域を高品質で圧縮したり、他の領域より先に表示したりできる。
著作権保護	第 3 者による勝手なアイドルコラージュといった不正加工をかなり困難にすることが出来る(Part8)。
	第 3 者による勝手な作品・写真の転載をかなり困難にすることができる(Part8)。
	第 3 者が勝手に作者本人であると偽ることをかなり困難にすることができる(Part8)。

計算機による画像処理や画像圧縮のテスト画像として広く世界中で利用されている Lena の画像を、従来の JPEG と JPEG2000 により、原画像の 1%のファイルサイズにエンコードしたものを図 6 に示す。特に拡大された目の部分をみると JPEG ではかなりブロックノイズが目立つのに対して、JPEG2000 ではなめらかである。また、JPEG では背景に縞状のノイズが現れているが、JPEG2000 では出ていない。

次に、ROI(Regions Of Interest : JPEG2000, Part1, Annex H)を用いてエンコードした画像を図 7 に示す。図 7 では、人物の顔の部分が高品質にエンコードされており、この領域の表示や伝送を先に行うことで閲覧者に対するストレスを小さくしたり、重要な部分の画質を保つことができる。ただし、ROI を用いない場合と圧縮率が同じであれば ROI を用いる方が ROI 以外の部分の画質が劣化する。



JPEG　　JPEG2000

図 6 JPEG と JPEG2000 でエンコードした画像の比較(圧縮率 1%)

図 7 ROI を用いてエンコードした画像

ここまで述べてきたように(表 6, 図 6, 図 7)、JPEG2000 では従来の JPEG と比較して優れた点が多く、近い将来 JPEG の後継として広く世界中で利用されていくことに疑う余地はない。

JPEG2000 には、機能や内容に応じて、いくつかのファイルフォーマットがある。JPEG2000 ファイルフォーマットの拡張子の一覧を表 7 に示す。

表 7 JPEG2000 の拡張子

拡張子	内容
.j2k	JPEG2000 の事実上総合標準拡張子
.pgx	立証モデル用のデータ。プロトタイプであって正式なものではない。
.jpc, .j2c	ISO15444-1(Part1, Annex A)。コードストリーム部のみで構成される、JPEG2000 のデータ。
.jp2	ISO15444-1(Part1, Annex I)。JPEG2000 データファイル拡張子。JP2 ヘッダ部とコードストリーム部で構成される。静止画 JPEG2000 はこのファイルにすべきであると JPEG 企画委員会では主張している。
.jpx	ISO15444-2(Part2)。JPEG2000 データファイル拡張子。JPX ヘッダ部とコードストリーム部で構成される。JP2 から機能強化されている。
.mj2	ISO15444-3(Part3)で定められた Motion JPEG2000 ファイル拡張子。MJ2 ヘッダ部とコードストリーム部で構成される。動画で音声も付加できる。
.jpm	ISO15444-6(Part6)で定められた JPEG2000 データファイル拡張子。JPM ヘッダ部とコードストリーム部で構成される。JPX から機能拡張したもので、レイヤ（層）画像、複合圧縮が可能になっている。
.j3d	ISO15444-10(Part10)で定められる JPEG2000 データファイル拡張子。J3D ヘッダ部とコードストリーム部で構成される。3 次元画像を扱うことができる。

JPEG2000 には優れた点が多いが、必ずしも万能ではない。例えば、

- JPEG2000 は圧縮率を高くしてもブロックノイズやモスキートノイズは生じないが、反面、画像がぼんやりする。
- JPEG2000 は写真グラデーションが効いた画像には向くが、スキャナで取り込んだ鉛筆による下書きなどに対して圧縮率を高く設定すると線がぼやけてしまう。
- JPEG2000 はかなり処理の負荷が大きい（特に可逆圧縮）。

といった点があり、状況や画像の特性に応じて使い分ける必要がある。

3.2. JPEG2000 のアルゴリズム

JPEG2000 では図 8 のように Reference grid と呼ばれる座標系を定義し、その上に画像を配置する。Reference grid の座標(0, 0)と画像の左上のピクセル(画素)の座標(0, 0)は一般的には同一でよい。

次に、画像を Reference grid の(0, 0)から、横方向に XTsiz ピクセル、縦方向に YTsiz ピクセルの大きさの、タイル(tile)と呼ばれる複数の領域に分割する。ここで、タイルの総数を numtiles とする。そして、図 8 のように、左上のタイルからラスター走査順に、0 から numtiles-1 までの番号(tileno)をつける。タイルの大きさ(XTsiz, YTsiz)はエンコード時にユーザーが指定する。JPEG2000 では、このタイルをひとつの単位としてエンコード・デコードを行う。また、画像全体を単一のタイルとしてエンコードしてもよい。

各タイルには、通常、カラー画像であれば RGB または YCbCr の 3 色のコンポーネント(colour component)により構成され、グレースケール画像であれば輝度(Y)などの 1 つのコンポーネントのみで構成されている。つまり、カラー画像は 3 枚の画像を重なり合わせたような形で見える。フルカラー画像での 1 ピクセルの 1 コンポーネントは通常、8 bits である。

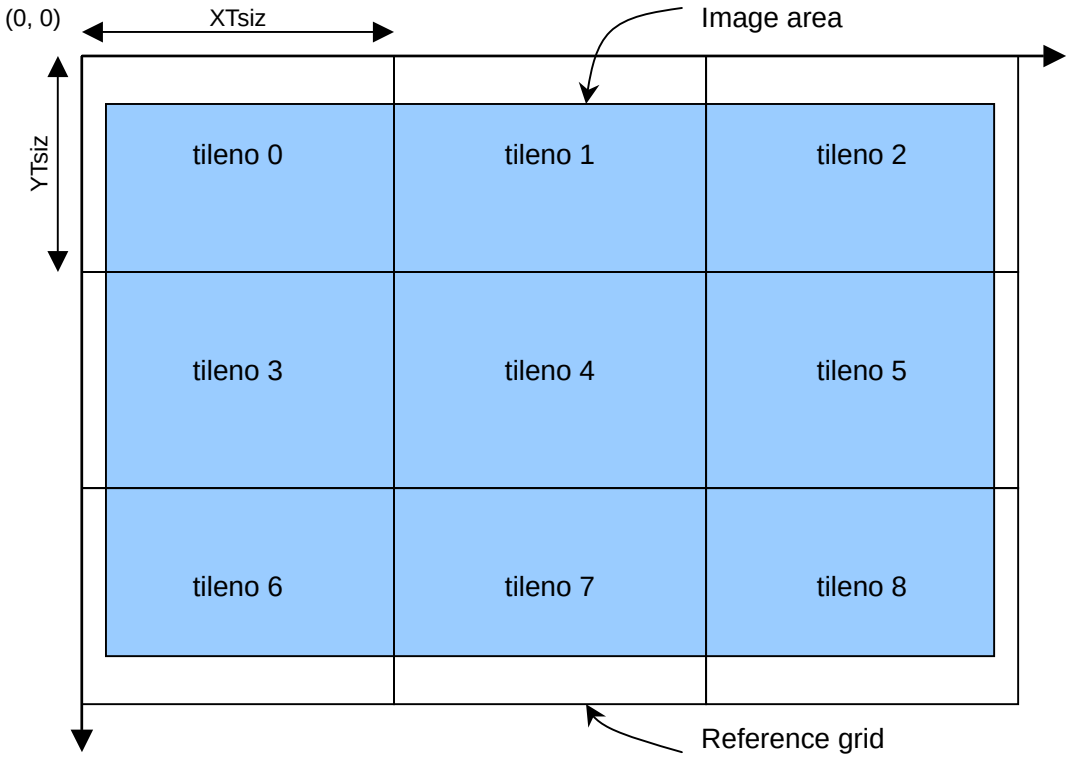


図 8 画像のタイルへの分割

画像のタイルへの分割が終わると、各タイルひとつひとつを、それぞれ以下の順でエンコードを行う。

1. DC レベルシフト ‥‥‥‥‥DLS : DC level shifting
2. コンポーネント間変換‥‥‥MCT : Multi-Component Transform
3. 離散ウェーブレット変換‥‥DWT : Discrete Wavelet Transform
4. 量子化 ‥‥‥‥‥‥‥‥‥‥QNT : Quantization
5. エブコット ‥‥‥‥‥‥‥‥EBCOT : Embedded Block Coding with Optimized Truncation

さらに、不可逆圧縮(lossy)モードでは、4, 5 において圧縮率の調整を行う。また、各タイルは独立してエンコードを行うことができる。

次に、1. ~ 5.の各処理について紹介する。詳細は ISO/IEC 15444 (ITU-T Rec. T.800)または 4.1.にて後述する JasPer[11]付属のドキュメントを参照のこと。

- DC レベルシフト (DLS)
  各コンポーネント(例: R, G, B)の信号値が正の値(符号無し整数)の場合、ダイナミックレンジの半分の値を減算するレベルシフトを行う。例えば、8 bits の場合、0～255 の値を、-128～127 にシフトする。

- コンポーネント間変換 (MCT)
  入力 RGB 信号を輝度と色差成分から成る YCbCr 色空間(color space)へ変換する。可逆圧縮の場合は変換式の整数演算の RCT(Reversible Color Transform)を用いる。また、不可逆圧縮の場合は変換式が実数演算の ICT(Irreversible Color Transform)を用いることもできる。

- 離散ウェーブレット変換 (DWT)
  PR UMDFB(Perfect-Reconstruction Uniformly-Maximally-Decimated Filter Bank)を用いて YCbCr の信号をそれぞれ周波数成分に変換する。UMDFB とは複数のフィルタ関数(filter transfer functions)、量子化演算子(quantization operators)、ゲイン(gains)を要素として構成されるフィルタバンクである。コンポーネント間変換のときと同様に整数演算（integer-to-integer, 可逆）と実数演算（real-to-real, 不可逆）とで UMDFB の要素の係数など、異なったものが用いられ、整数演算は 5/3 フィルタ、実数演算は 9/7 フィルタと呼ばれる。UMDFB は 1 次元の信号を扱うが(1-D UMDFB)、JPEG2000 エンコーディングの対象は静止画像で 2 次元である。そこで、1-D UMDFB を画像の水平・垂直方向、両方に適用することで 2 次元の変換を行える。これを 2-D UMDFB という。

ウェーブレット変換は、タイルコンポーネント(tile-component)に対して繰り返し行われる。2-D UMDFB の適用により、様々なサブバンド(subbands)が出力され、それらを、水平ローパス(lowpass)垂直ローパス(LL)、水平ローパス垂直ハイパス(highpass)(LH)、水平ハイパス垂直ローパス(HL)、水平ハイパス垂直ハイパス(HH)の 4 つに分類する。はじめの変換を R - 1 レベルとすると、それぞれ LL_{R-1}, LH_{R-1}, HL_{R-1}, HH_{R-1} というサブバンドが得られる。そして、更に LL_{R-1} を分析して $R-2$ レベルの LL_{R-2}, LH_{R-2}, HL_{R-2}, HH_{R-2} が得られる。これを LL_0 が得られるまで LL バンドの分析を繰り返す。各 LL はそのレベルでの解像度となり、このレベルのことをリゾリューションレベル(resolution level)という。全部で R 個のリゾリューションレベルが生成され、もっとも粗い解像度は 0 レベルであり、最も良い解像度は $R-1$ レベルとなる。R の値はユーザーがエンコード時に指定することもできる。ここまで説明したタイルコンポーネントデータの離散ウェーブレット変換により生成されるサブバンドの関係と構造を図 9 に示す。

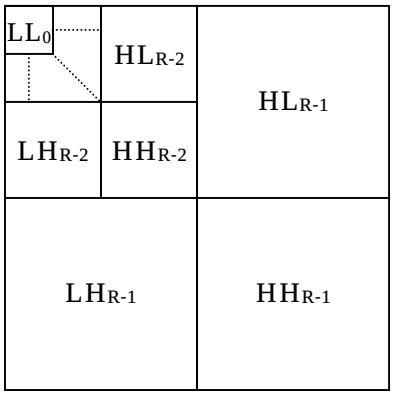


図 9 離散ウェーブレット変換により生成されるサブバンド構造

- 量子化 (QNT)

離散ウェーブレット変換により生成されたサブバンド信号を、量子化ステップサイズ(quantizer step size)で割ることにより、ダイナミックレンジの削減を行う。量子化の処理においても、整数モード(integer mode)と実数モード(real mode)があり、可逆圧縮では整数モード、不可逆圧縮では実数モードが用いられる。ここで、整数モードでの量子化ステップサイズは常に 1 であり、実際には量子化の処理はバイパスする。つまり、実数モードの場合のみ量子化が行われる。量子化ステップサイズは各サブバンド（タイル・コンポーネント・解像度の組み合わせ）に対して別々の値を用いることもできる。

- エブコット (EBCOT)

  まず、ウェーブレット変換と量子化により得られた各サブバンドを、さらにコードブロック(code block)と呼ばれる複数の領域に分割する。各コードブロックは以後独立して符号化できる。コードブロックはビットプレーン(bit-plane)として扱われ、MSB(Most Significant Bit)プレーンから LSB(Least Significant Bit)プレーンの順で符号化が行われる。例えば 8 bits のサンプルの場合、コードブロックは 8 枚のビットプレーンから成り、MSB プレーンは符号ビットプレーン(Significant bit-plane)となる。Significance propagation pass, Magnitude refinement pass, Cleanup pass という 3 つのパスによりビットプレーンを処理したものを、エントロピー符号化することによってビット列の圧縮を行う。エントロピー符号化では適応二進算術符号化(adaptive binary arithmetic coder)の MQ コーダーを用いる。また、不可逆圧縮では、ビット切り捨てによるポスト量子化により圧縮率の制御を行う。更に、ビット列をパケット(packet)と呼ばれる単位で、出力ストリームに書き込む。

ここまでをまとめると、逐次での JPEG2000 エンコード処理の流れは、図 10 のようになる。

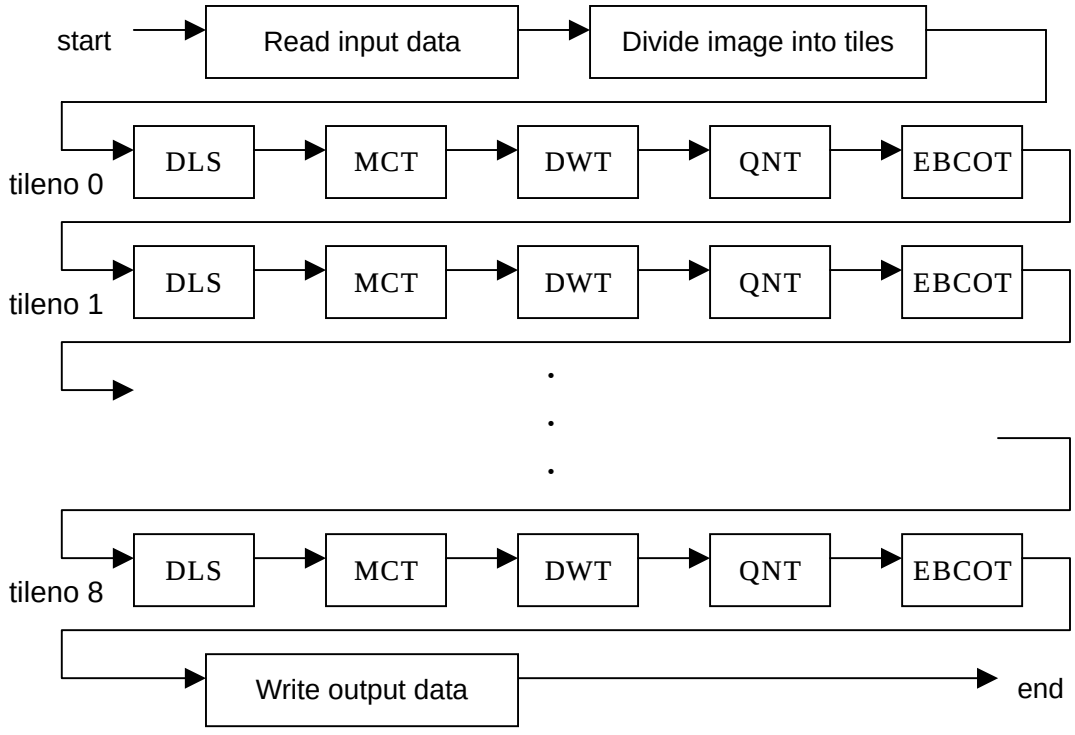


図 10 逐次での JPEG2000 エンコードの流れ

4. PC クラスタ上での OpenMP による JPEG2000 の並列化

JPEG2000 の高圧縮率かつ高品質という優れた点は、データ伝送時間の短小、データの保存領域の縮小という利点を生む。しかし逆に、画像データの圧縮に要する処理負荷が大きいという側面は、圧縮時間が多くかかるという欠点になる。

医療や出版などの特に高解像度の画像が必要な分野では、この圧縮にかかる処理負荷による圧縮時間の増大がストレスとなることが考えられる。そこで、PC クラスタによる並列処理により、高速に画像の圧縮を行い、圧縮時間を高速化することで、ストレスの少ない画像のやりとりが可能になる。

4.1. 実装方法

JPEG2000 では、JasPer[11]という C 言語のリファレンスソフトウェア実装(JPEG2000 Part 5)が公開されている。今回は、この JasPer(バージョン 1.600.0)をベースとして、SCore PC クラスタの OpenMP 環境である Omni-SCASH 環境への移植を行い、並列化を行った。

4.2. ソフトウェア分散共有メモリの動的確保と解放

通常、C 言語で書かれた、逐次環境で動作するプログラムは、メモリの動的確保と解放のために、malloc(), free()関数を利用している。

Omni-SCASH 環境でも、ソフトウェア分散共有メモリ(DSM)を動的に確保する ompsm_galloc()という関数が用意されている。しかし、ompsm_galloc()で確保したメモリを解放する、free()にあたる関数は用意されていない。JasPer を Omni-SCASH 環境へ移植する際、このことが大きな問題となった。

そこで、DSM の動的確保、解放の機能を有する、omscmmm_malloc(), omscmmm_free()という関数を ompsm_galloc()を利用して作成することでこの問題の解決を行った。その仕組みを図 11 に示す。この omscmmm_malloc(), omscmmm_free()は、初めて omscmmm_malloc()が呼び出されたときに、その SCore クラスタシステムの Omni-SCASH で利用可能な最大のサイズ(コンパイル時に指定)の DSM を確保する。そして以後は、最初に確保した DSM 領域の中で動的確保、解放を行う。このようにすることで、JasPer などのアプリケーション側からみれば、DSM の動的確保と解放が実現可能となった。

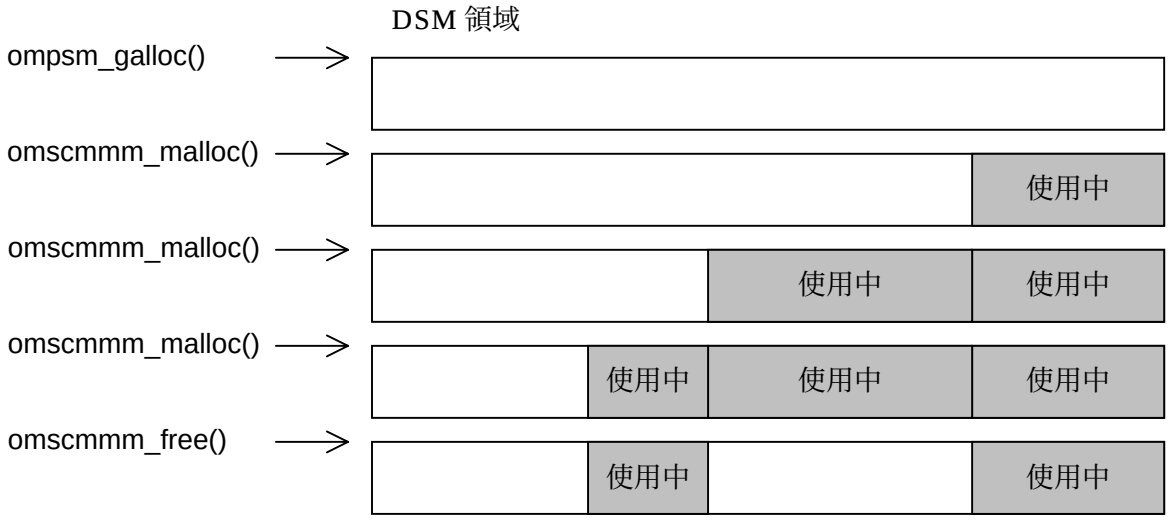


図 11 omscmmm_malloc()と omscmmm_free()

また、JasPer は移植性を考慮して設計されており、メモリの動的確保と解放はすべて図 12 のような jas_malloc()と jas_free()というラッパー関数を介して行われている。これらの関数の中身を omscmmm_malloc(), omscmmm_free()に置き換えることで、ソフトウェア分散共有メモリの動的確保と解放を実現した。

```
void *jas_malloc(size_t size)
{
   return malloc(size);
}

void jas_free(void *ptr)
{
   free(ptr)
}
```

```
void *jas_malloc(size_t size)
{
   return omscmmm_malloc(size);
}

void jas_free(void *ptr)
{
   omscmmm_free(ptr)
}
```

図 12 jas_malloc()と jas_free()

4.3. OpenMP による並列化方法

図 8,図 10 より、各タイルは独立してエンコードすることが可能である。よって、並列化を行うには、各タイルのエンコード処理を、OpenMP の#pragma omp parallel for 指示文によりそれぞれのスレッドに分担させればよい。

つまり、エンコードの処理は図 13 のように記述される。本研究では、繰り返しの割り当て方法を指定する schedule 指示節を用いて、schedule(runtime)とすることで、実行時にユーザーが、環境変数 OMP_SCHEDULE により割り当て方法を指定できるようにした。

```
入力ファイルを読み込む
画像をタイルへ分割する
#pragma omp parallel for schedule(runtime)
for (tileno = 0; tileno < numtiles; tileno++) {
   タイル番号がtileno のタイルをエンコードする
}
出力ファイルを書き出す
```

図 13 エンコード部の OpenMP 指示文による並列化

以上のように並列化を行ったとき、JPEG2000 のエンコード処理の流れは、図 14 のようになる。

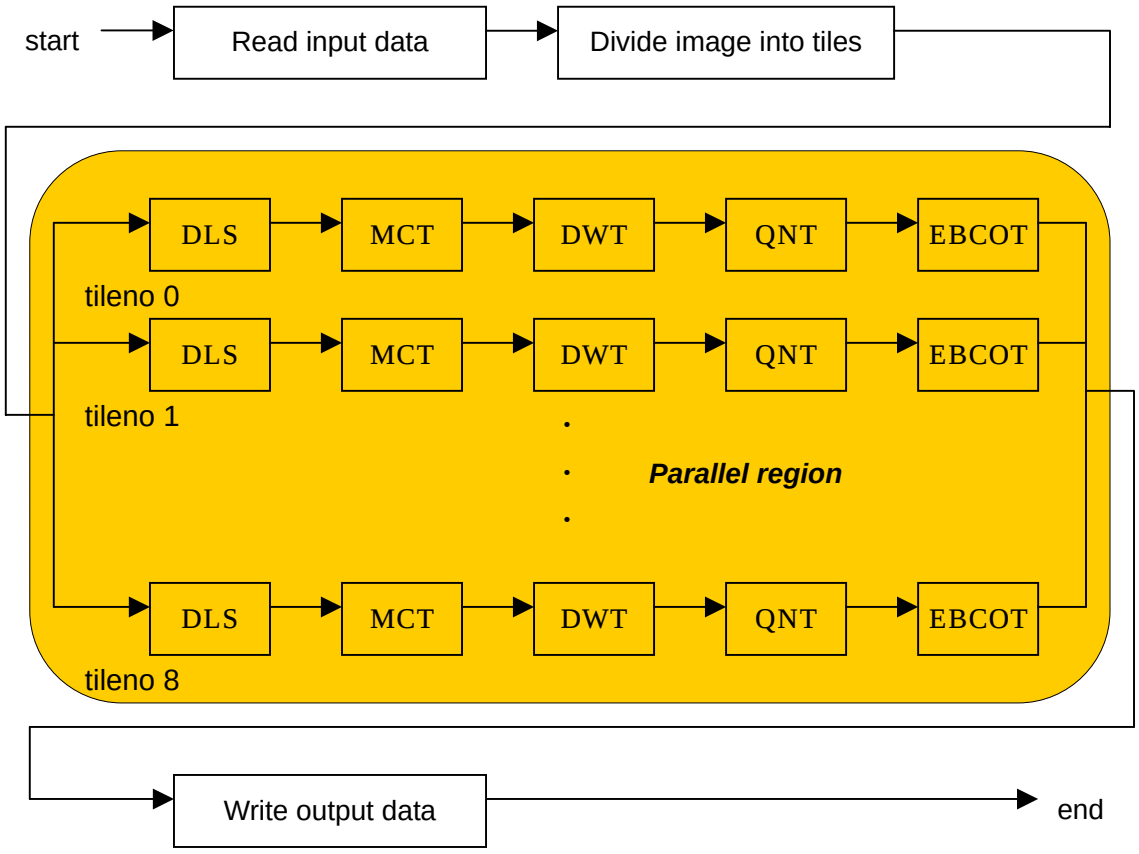


図 14 並列化した JPEG2000 エンコードの流れ

図 14 のように、各タイルの処理は並列リージョン中で各スレッドに分担して処理される。ただし、ファイル読み込み、タイルへの分割、出力ファイルの書き込みは並列化できないため、マスタースレッドのみが逐次で実行する。

5. 実行結果と考察

5.1. 実行結果

研究室の SCore PC クラスタ(図 1,表 1,表 2)において、表 8 に示す条件で実行し、実行時間を測定した。

表 8 実行条件

入力画像	大きさ 1960×1420 の画像 (PPM 形式、カラー画像、ファイルサイズ 8,643,617bytes)
圧縮モード	可逆(lossless)圧縮
タイルサイズ	50×50, 100×100, 150×150, 200×200, 250×250
OMP_SCHEDULE	static, static,1, dynamic,1

実行に用いた入力画像を図 15 に示す。

図 15 実行に用いた画像

5.1.1. タイルサイズの違いによる実行結果

はじめに、OMP_SCHEDULE を static に固定し、タイルサイズを 50×50, 100×100, 150×150, 200×200, 250×250 としたときの出力された JP2 ファイルサイズと、入力 PPM ファイルサイズに対する圧縮率を表 9, 図 16 に示す。

また、上述のようにタイルサイズを変えて、スレッド数 1, 2, 4, 8, 16 のときの実行時間と、スレッド数 1 に対する速度向上比を表 10, 図 18 に示す。実行時間には、ファイルの入出力処理の時間は含まれていない。

表 9 タイルサイズの違いによる出力 JP2 ファイルサイズ・圧縮率

タイルサイズ	JP2 ファイルサイズ[bytes]	JP2/PPM 圧縮率[%]
50×50	1,751,754	20.27
100×100	1,519,118	17.58
150×150	1,461,687	16.91
200×200	1,434,243	16.59
250×250	1,416,759	16.39

表 10 タイルサイズの違いによる実行時間と速度向上比(OMP_SCHEDULE:static)

タイルサイズ タイル数		ノード数				
		1	2	4	8	16
50×50	実行時間[sec]	47.28	25.02	13.43	7.42	4.33
1200	速度向上比	1.00	1.89	3.53	6.39	10.95
100×100	実行時間[sec]	29.08	15.76	8.83	5.13	6.39
300	速度向上比	1.00	1.85	3.30	5.66	9.13
150×150	実行時間[sec]	24.53	13.75	7.96	4.78	3.02
140	速度向上比	1.00	1.78	3.08	5.13	8.14
200×200	実行時間[sec]	23.07	13.98	8.14	4.71	2.99
80	速度向上比	1.00	1.65	2.83	4.90	7.71
250×250	実行時間[sec]	22.50	13.06	7.66	4.65	2.92
48	速度向上比	1.00	1.72	2.94	4.84	7.70

5.1.2. OMP_SCHEDULE の違いによる実行結果

次に、タイルサイズを 50×50 に固定し、環境変数 OMP_SCHEDULE を、static, static,1, dynamic,1 としたときのスレッド数 1, 2, 4, 8, 16 での実行時間とスレッド数 1 に対する速度向上比を表 11, 図 18 に示す。

表 11 OMP_SCHEDULE の違いによる実行時間と速度向上比(タイルサイズ 50×50)

OMP_SCHEDULE		ノード数				
		1	2	4	8	16
static	実行時間[sec]	47.38	25.02	13.43	7.42	4.33
	速度向上比	1.00	1.89	3.53	6.39	10.95
static,1	実行時間[sec]	47.46	25.94	14.43	8.56	5.49
	速度向上比	1.00	1.83	3.29	5.55	8.64
dynamic,1	実行時間[sec]	47.33	26.67	14.98	8.89	5.49
	速度向上比	1.00	1.77	3.16	5.32	8.62

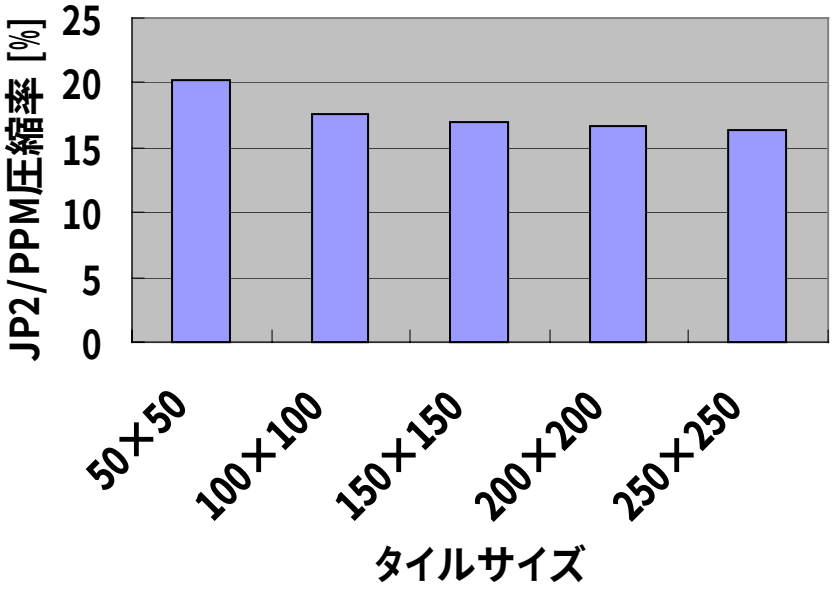


図 16 タイルサイズの違いによる JP2/PPM 圧縮率

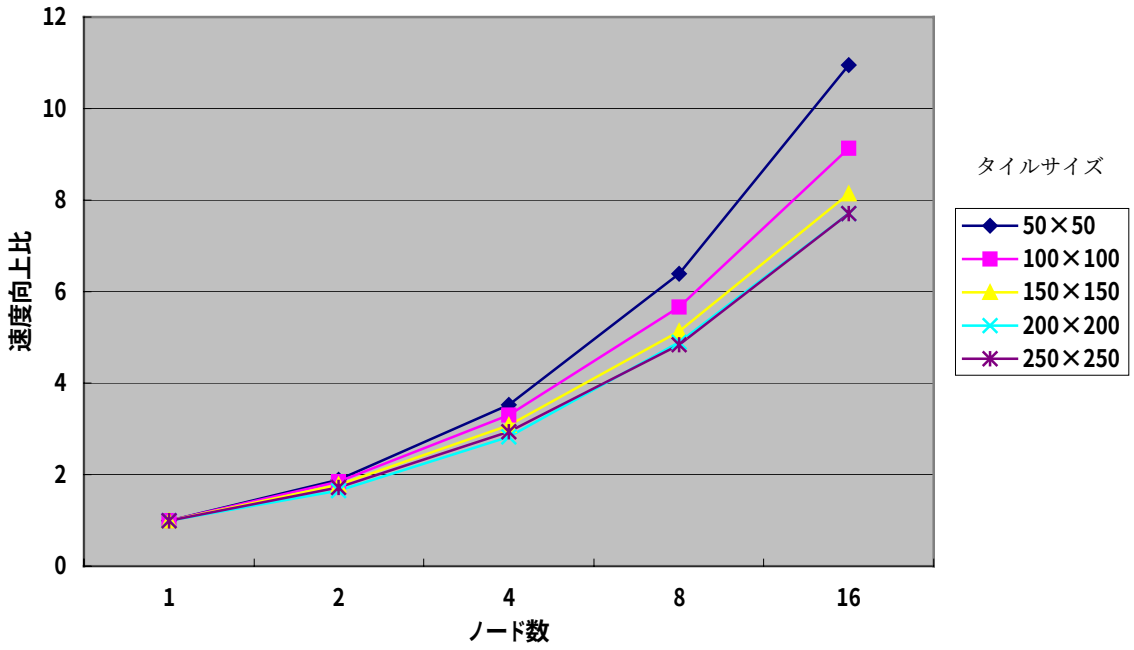


図 17 タイルサイズの違いによる実行結果（速度向上比）

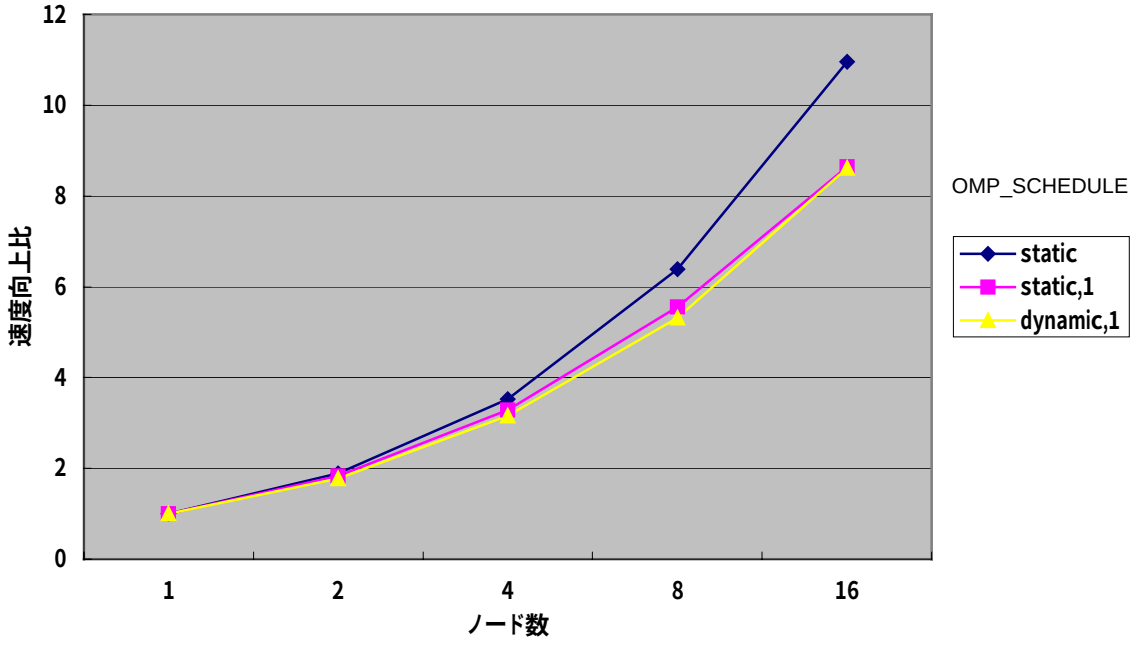


図 18 OMP_SCHEDULE の違いによる実行結果（速度向上比）

5.2. 考察

表 10 より、タイルサイズが大きいほうが実行時間は短くなり、逆にタイルサイズが小さいほうが速度向上比が良くなっている。また、表 9, 図 16 より、タイルサイズが大きいほうが圧縮率が良い。

この原因は、タイルサイズが小さいとタイル数が多くなり、マーカー(データとデータの間の目印のようなもの)の挿入などによる処理時間が大きくなるためと思われる。しかし、タイルサイズを小さくすることによりエラーの回復性は良くなる。ここでいうエラーの回復性とは、伝送などでファイルの破損があると、タイルサイズが小さいほど悪影響を受ける画像の領域が小さくなるということである。

つまり、タイルサイズの大小は、「実行時間と圧縮率 vs 速度向上比とエラー回復性」のトレードオフとなり、どちらが良いとは一般的には言えず、ユーザーがそのときの状況により判断するものである。

また、表 11, 図 18 より、OMP_SCHEDULE が static のときが、static,1、dynamic,1 よりも良い結果となった。これは、各タイルの処理の負荷のばらつきが少ないことが原因である。ただし、JPEG2000 で規定されている ROI(Region Of Interest)を利用した場合や、不可逆(lossy)でサブバンドごとに量子化ステップサイズを変えたときは、dynamic,1 や static,1 の方が良い結果が得られることもあるものと思われる。

さらに、今回は、速度向上比がもっとも良い結果のもので、タイルサイズ 50×50、OMP_SCHEDULE が static の 16 ノードで約 11 倍となった。今回のような JPEG2000 の並列化では、タイルの処理がそれぞれ完全に独立して行え、また負荷のばらつきも小さいため、本来では 16 ノードで 16 倍の速度向上比というように、線形に向上していくはずである。

この原因として、ほとんどすべてのデータをソフトウェア分散共有メモリ上に置いており、共有メモリアクセスによる通信のオーバーヘッドがかなり大きくなっているということが考えられる。よって、並列リージョン中は分散共有メモリに置く必要のない変数やメモリ領域を、2.2.でふれた private などの指示節や、通常の malloc(), free()により、スレッドのローカルメモリを利用するようプログラムを改変することで、更なるエンコード時間の短縮と速度向上が得られるものと考えられる。

本研究ではタイルのエンコードを各スレッドへ分担させるという手法により JPEG2000 の並列化を行ったが、3.2.で説明した DC レベルシフトやコンポーネント間変換、コードブロックの符号化も OpenMP の#pragma omp parallel for 指示文での並列化が可能である。この方法を実装し、タイル分割の方法と比較検討する必要がある。

6. おわりに

本研究では、PC クラスタ上の OpenMP により、JPEG2000 エンコーダの並列化を行った。またその過程において、SCore PC クラスタの分散共有メモリを動的に確保・解放を行う、omscmmm_malloc(), omscmmm_free()関数の設計と実装を行った。

JPEG2000 の並列化はタイルのエンコードをスレッドに分担させる方法を用い、5 章において実際に、タイルサイズ、スレッドへの割り当て方法を変化させてエンコードを行い、実行時間を計測した。その結果、1 ノードでの実行に対して、16 ノードで約 11 倍の速度向上を得ることができた。

今後の課題としては、まず、エンコーダの更なる高速化とデコーダーの並列化が挙げられる。また、本研究で用いているタイルをスレッドに割り当てる方法とは別の、5.2.で述べたタイルのエンコードの各ステージをそれぞれ並列化する方法との比較を行う必要がある。さらに、Motion-JPEG2000 への対応が挙げられ、高速化を行うことで、リアルタイムでのエンコード、配信を行うシステムの実現などが考えられる。

謝辞

本研究の機会を与えてくださり、ご指導を頂きました山崎勝弘教授、小柳滋教授に深く感謝致します。また、本研究に関して貴重な助言や励ましを頂きました Tran So Cong 氏、池田修久氏、及び研究室の皆様に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、貴重な助言を頂き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] 新情報処理開発機構(RWCP) : http://www.rwcp.or.jp/

[3] Myricom, Inc. : http://www.myri.com/

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

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

[6] 佐藤 三久 : OpenMP, JSPP'99 OpenMP チュートリアル資料, 1999

[7] 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

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

[9] JPEG : http://www.jpeg.org/

[10] ISO/IEC 15444-1: Information technology - JPEG2000 image coding system - Part 1: Core coding system, 2000

[11] JasPer : http://www.ece.ubc.ca/~mdadams/jasper/

[12] 小野 定康, 鈴木 順司 : わかりやすい JPEG/MPEG の技術, オーム社, 2001

[13] 湯浅 太一, 安藤 通晃, 中田 登志之 : bit 別冊 初めての並列プログラミング, 共立出版, 1998

[14] 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

[15] 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

[16] 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

[17] 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

[18] 池上 広済 : PC クラスタ上での OpenMP による JPEG 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

1. omscmmm_malloc(), omscmmm_free() ソースコード

omscmmm_malloc.h

```
#ifndef OMSCMMM_MALLOC_H
#define OMSCMMM_MALLOC_H
/* prototypes */
void *omscmmm_malloc(unsigned int n_bytes);
void   omscmmm_free(void *tofree_p);
void *omscmmm_realloc(void *ptr, size_t n_bytes);
void   omscmmm_print_memusage(void);
void *omscmmm_malloc_local(size_t n_bytes);
void   omscmmm_free_local(void *ptr);
#endif
```

omscmmm_malloc.c

```
#include <stdio.h>
#include <stdlib.h>
#ifdef __OMNI_SCASH__
#include <omp.h>
#define XMALLOC(x)   ompsm_galloc((x), OMNI_DEST_NONE, 0)
#else
#define XMALLOC(x)   malloc(x)
#endif
#include "jasper/omscmmm_malloc.h"
#include "jasper/jas_debug.h"

typedef double omscmmm_malloc_align_t; /* 割付単位の大きさ */

union omscmmm_malloc_header {
   struct {
      union omscmmm_malloc_header *ptr;/* 次の空きブロック */
      unsigned int size;/* このブロックの単位数 */
   } s;
   omscmmm_malloc_align_t x;/* ブロックの整合を強制 */
};
typedef union omscmmm_malloc_header omscmmm_malloc_header_t;

static omscmmm_malloc_header_t   omscmmm_malloc_base;
static omscmmm_malloc_header_t *omscmmm_malloc_free_p = NULL;
/* 空きブロックリストの先頭 */
```

```
/* for debug */
static int ttl_n_units = 0; /* malloc 合計サイズ */
static int cur_n_units = 0; /* 現在 malloc されてるサイズ */
static int max_n_units = 0; /* malloc された最大値 */

#define OMSCMMM_NUMALLOCSIZE 20*1024*1024 /* 要求する最小単位数 */

static
omscmmm_malloc_header_t *
omscmmm_moregalloc(unsigned int req_n_units)
{
   omscmmm_malloc_header_t *up;
   unsigned int n_units = 0;
   static int xmalloced_flag = 0;

   if(xmalloced_flag) {
      printf("ERROR: moregalloc CALLED TWICE¥n");
      exit(1);
   }
   if(req_n_units < OMSCMMM_NUMALLOCSIZE) {
      n_units = OMSCMMM_NUMALLOCSIZE; /* 最低これだけ確保する */
   } else {
      n_units = req_n_units;
   }
   if((up=(omscmmm_malloc_header_t*)
            XMALLOC(n_units*sizeof(omscmmm_malloc_header_t))) == NULL) {
      return NULL; /* 確保失敗 */
   } else {
      xmalloced_flag = 1;
      up->s.size = n_units;
      omscmmm_free((void *)(up + 1)); /* 空きブロックリストに繋ぐ */
      return omscmmm_malloc_free_p; /* 空きブロックリストの先頭を返す */
   }
   return NULL;
}

void *
omscmmm_malloc(unsigned int n_bytes)
{
   omscmmm_malloc_header_t *p, *prev_p;
   unsigned int n_units = 0;

   /* 確保する単位数を決める */
   n_units=(n_bytes+sizeof(omscmmm_malloc_header_t)-1)
            /sizeof(omscmmm_malloc_header_t) + 1;
   if((prev_p = omscmmm_malloc_free_p) == NULL) {
      /* はじめての呼び出し(空きブロックリストがまだない) */
      omscmmm_malloc_base.s.ptr=omscmmm_malloc_free_p=prev_p
               =&omscmmm_malloc_base;
      omscmmm_malloc_base.s.size = 0;
```

```
    }
    ttl_n_units += n_units;
    cur_n_units += n_units;
    max_n_units = (cur_n_units > max_n_units) ? cur_n_units : max_n_units;
    JAS_DBGLOG(5,("::::::total:%d,max:%d,cur:%d::::::¥n",
                  ttl_n_units*sizeof(omscmmm_malloc_header_t),
                  max_n_units*sizeof(omscmmm_malloc_header_t),
                  cur_n_units*sizeof(omscmmm_malloc_header_t)));
    /* 大きさの合う空きブロックリストを探す(first-fit) */
    for(p = prev_p->s.ptr; ; prev_p = p, p = p->s.ptr) {
      if(p->s.size >= n_units) { /* 十分大きい */
        if(p->s.size == n_units) { /* ぴったり */
          prev_p->s.ptr = p->s.ptr;
        } else { /* 後尾を割り当てる */
          p->s.size -= n_units;
          p += p->s.size;
          p->s.size = n_units;
        }
        omscmmm_malloc_free_p = prev_p;
        return (void *)(p + 1); /* データ領域へのポインタを返す */
      }
      if(p == omscmmm_malloc_free_p) { /* 要求サイズを満たす空きブロックが無い */
        if((p = omscmmm_moregalloc(n_units)) == NULL) {
          printf("ERROR: CANNOT ALLOCATE MEMORY.¥n");
          exit(1);
          return NULL; /* 確保失敗 */
        }
      }
    }
}

void
omscmmm_free(void *tofree_p)
{
    omscmmm_malloc_header_t *bp, *p;
    unsigned int n_units = 0;

    /* このブロックのヘッダ部を指す */
    bp = (omscmmm_malloc_header_t *)tofree_p - 1;
    n_units = (bp->s.size < OMSCMMM_NUMALLOCSIZE) ? bp->s.size : 0;
    cur_n_units -= n_units;
    for(p = omscmmm_malloc_free_p; !(bp > p && bp < p->s.ptr); p = p->s.ptr) {
      if(p >= p->s.ptr && (bp > p || bp < p->s.ptr)) {
        break; /* 領域の始めあるいは終わりの解放ブロック */
      }
    }
    if(bp + bp->s.size == p->s.ptr) { /* 後の空きブロックへ結合 */
      bp->s.size += p->s.ptr->s.size;
      bp->s.ptr     = p->s.ptr->s.ptr;
    } else {
```

```
    bp->s.ptr = p->s.ptr;
  }
  if(p + p->s.size == bp) {              /* 前の空きブロックへ結合 */
    p->s.size += bp->s.size;
    p->s.ptr  = bp->s.ptr;
  } else {
    p->s.ptr = bp;
  }
  omscmmm_malloc_free_p = p; /* 空きブロックリストの先頭を返す */
}

void *
omscmmm_realloc(void *ptr, size_t n_bytes)
{
  omscmmm_malloc_header_t *pre_p, *p;
  unsigned int pre_n_units, new_n_units, cpy_n_units;

  pre_p = (omscmmm_malloc_header_t *)ptr - 1; /* 元のブロックのヘッダ部を指す */
  pre_n_units = pre_p->s.size;                /* 元のブロックのサイズを得る */
  /* 新しいブロック単位数を求める */
  new_n_units=(n_bytes+sizeof(omscmmm_malloc_header_t)-1)
                /sizeof(omscmmm_malloc_header_t)+1;
  /* コピーするブロック単位数を求める */
  cpy_n_units = (pre_n_units < new_n_units) ? pre_n_units : new_n_units;
  /* 新しいメモリを確保 */
  if((p = omscmmm_malloc(n_bytes)) == NULL) {
    printf("ERROR: CANNOT ALLOCATE MEMORY.¥n");
    exit(1);
    return NULL; /* 確保失敗 */
  }
  /* メモリをコピー */
  memcpy(p, ptr, cpy_n_units * sizeof(omscmmm_malloc_header_t));
  /* 元のメモリブロックを解放 */
  omscmmm_free(ptr);
  return (void *)p;
}

void
omscmmm_print_memusage(void)
{
  printf("::::::total:%d, max:%d, cur:%d::::::¥n",
   ttl_n_units*sizeof(omscmmm_malloc_header_t),
   max_n_units*sizeof(omscmmm_malloc_header_t),
   cur_n_units*sizeof(omscmmm_malloc_header_t));
}
```

2. エンコーダ並列化部 ソースコード

jpc_enc_encodemainbody()

```
static int jpc_enc_encodemainbody(jpc_enc_t *enc)
{
    int tileno;
    int numtiles;
    jpc_enc_t **encs;
    double sec;

    numtiles = enc->cp->numtiles;
    if(!(encs = (jpc_enc_t **)jas_malloc(numtiles * sizeof(jpc_enc_t)))) {
        ERROR(); return -1;
    }
    for(tileno = 0; tileno < numtiles; tileno++) {
        if(!(encs[tileno] = jpc_enc_copy(enc))) {
            ERROR(); return -1;
        }
        if (!(encs[tileno]->tmpstream = jas_stream_memopen(0, 0))) {
            ERROR(); return -1;
        }
    }
    sec = second();

#pragma omp parallel for schedule(runtime)
    for (tileno = 0; tileno < numtiles; ++tileno) {
        /* int tilex; */
        /* int tiley; */
        int i;
        jpc_sot_t *sot;
        jpc_enc_tcmpt_t *comp;
        jpc_enc_tcmpt_t *endcomps;
        jpc_enc_band_t *band;
        jpc_enc_band_t *endbands;
        jpc_enc_rlvl_t *lvl;
        int rlvlno;
        jpc_qcc_t *qcc;
        jpc_cod_t *cod;
        int adjust;
        int j;
        int absbandno;
        long numbytes;
        long tilehdrlen;
        long tilelen;
        jpc_enc_tile_t *tile;
        jpc_enc_cp_t *cp;
        double rho;
        uint_fast16_t lyrno;
```

```
    uint_fast16_t cmptno;
    int samestepsizes;
    jpc_enc_ccp_t *ccps;
    jpc_enc_tccp_t *tccp;
    int bandno;
    uint_fast32_t x;
    uint_fast32_t y;
    int mingbits;
    int actualnumbps;
    jpc_fix_t mxmag;
    jpc_fix_t mag;
    int numgbits;

    cp = encs[tileno]->cp;
    /* Avoid compile warnings. */
    numbytes = 0;
#if 0
    tilex = tileno % cp->numhtiles;
    tiley = tileno / cp->numhtiles;
#endif
    if (!(encs[tileno]->curtile
          =jpc_enc_tile_create(encs[tileno]->cp,encs[tileno]->image, tileno))) {
       ERROR(); abort();
    }
    tile = encs[tileno]->curtile;
    if (jas_getdbglevel() >= 10) {
       jpc_enc_dump(encs[tileno]);
    }
    endcomps = &tile->tcmpts[tile->numtcmpts];
    for (cmptno = 0, comp = tile->tcmpts; cmptno < tile->numtcmpts; ++cmptno, ++comp) {
       if (!cp->ccps[cmptno].sgnd) {
          adjust = 1 << (cp->ccps[cmptno].prec - 1);
          for (i = 0; i < jas_matrix_numrows(comp->data); ++i) {
             for (j = 0; j < jas_matrix_numcols(comp->data); ++j) {
                *jas_matrix_getref(comp->data, i, j) -= adjust;
             }
          }
       }
    }
    if (!tile->intmode) {
       endcomps = &tile->tcmpts[tile->numtcmpts];
       for (comp = tile->tcmpts; comp != endcomps; ++comp) {
          jas_matrix_asl(comp->data, JPC_FIX_FRACBITS);
       }
    }
    switch (tile->mctid) {
    case JPC_MCT_RCT:
       assert(jas_image_numcmpts(encs[tileno]->image) == 3);
       jpc_rct(tile->tcmpts[0].data, tile->tcmpts[1].data,
          tile->tcmpts[2].data);
```

```
   break;
case JPC_MCT_ICT:
   assert(jas_image_numcmpts(encs[tileno]->image) == 3);
   jpc_ict(tile->tcmpts[0].data, tile->tcmpts[1].data,
     tile->tcmpts[2].data);
   break;
default:
   break;
}

for (i = 0; i < jas_image_numcmpts(encs[tileno]->image); ++i) {
   comp = &tile->tcmpts[i];
   jpc_tsfb_analyze(comp->tsfb, ((comp->qmfbid == JPC_COX_RFT)
                                 ? JPC_TSFB_RITIMODE : 0), comp->data);
}
endcomps = &tile->tcmpts[tile->numtcmpts];
for (cmptno = 0, comp = tile->tcmpts; comp != endcomps; ++cmptno, ++comp) {
   mingbits = 0;
   absbandno = 0;
   /* All bands must have a corresponding quantizer step size,
     even if they contain no samples and are never coded. */
   /* Some bands may not be hit by the loop below, so we must
      initialize all of the step sizes to a sane value. */
   memset(comp->stepsizes, 0, sizeof(comp->stepsizes));
   for (rlvlno = 0, lvl = comp->rlvls; rlvlno < comp->numrlvls; ++rlvlno, ++lvl) {
     if (!lvl->bands) {
        absbandno += rlvlno ? 3 : 1;
        continue;
     }
     endbands = &lvl->bands[lvl->numbands];
     for (band = lvl->bands; band != endbands; ++band) {
        if (!band->data) {
           ++absbandno;
           continue;
        }
        actualnumbps = 0;
        mxmag = 0;
        for (y = 0; y < jas_matrix_numrows(band->data); ++y) {
           for (x = 0; x < jas_matrix_numcols(band->data); ++x) {
              mag = abs(jas_matrix_get(band->data, y, x));
              if (mag > mxmag) {
                 mxmag = mag;
              }
           }
        }
        if (tile->intmode) {
           actualnumbps = jpc_firstone(mxmag) + 1;
        } else {
           actualnumbps = jpc_firstone(mxmag) + 1 - JPC_FIX_FRACBITS;
        }
```

```
            numgbits = actualnumbps - (cp->ccps[cmptno].prec - 1 +band->analgain);
            if (numgbits > mingbits) {
               mingbits = numgbits;
            }
            if (!tile->intmode) {
                  band->absstepsize=jpc_fix_div(jpc_inttofix(1<<(band->analgain+1)),
                                 band->synweight);
            } else {
               band->absstepsize = jpc_inttofix(1);
            }
            band->stepsize = jpc_abstorelstepsize(band->absstepsize,
                                 cp->ccps[cmptno].prec + band->analgain);
            band->numbps = cp->tccp.numgbits + JPC_QCX_GETEXPN(band->stepsize) - 1;
             if ((!tile->intmode) && band->data) {
               quantize(band->data, band->absstepsize);
            }
            comp->stepsizes[absbandno] = band->stepsize;
            ++absbandno;
         }
      }
      assert(JPC_FIX_FRACBITS >= JPC_NUMEXTRABITS);
      if (!tile->intmode) {
         jas_matrix_divpow2(comp->data, JPC_FIX_FRACBITS - JPC_NUMEXTRABITS);
      } else {
         jas_matrix_asl(comp->data, JPC_NUMEXTRABITS);
      }
   }
   if (mingbits > cp->tccp.numgbits) {
      printf( "error: too few guard bits (need at least %d)¥n", mingbits);
      ERROR();
      /* return -1; */
   }
   /* Write the tile header. */
   if (!(encs[tileno]->mrk = jpc_ms_create(JPC_MS_SOT))) {
      ERROR();
      /* return -1; */
   }
   sot = &encs[tileno]->mrk->parms.sot;
   sot->len = 0;
   sot->tileno = tileno;
   sot->partno = 0;
   sot->numparts = 1;
   if (jpc_putms(encs[tileno]->tmpstream, encs[tileno]->cstate, encs[tileno]->mrk)) {
      printf( "cannot write SOT marker¥n");
      ERROR();
      /* return -1; */
   }
   jpc_ms_destroy(encs[tileno]->mrk);
   encs[tileno]->mrk = 0;
   tccp = &cp->tccp;
```

```
for (cmptno = 0; cmptno < cp->numcmpts; ++cmptno) {
   comp = &tile->tcmpts[cmptno];
   if (comp->numrlvls != tccp->maxrlvls) {
      if (!(encs[tileno]->mrk = jpc_ms_create(JPC_MS_COD))) {
         ERROR();
         /* return -1; */
      }
      /* XXX = this is not really correct. we are using comp #0's precint sizes
        and other characteristics */
       comp = &tile->tcmpts[0];
       cod = &encs[tileno]->mrk->parms.cod;
       cod->compparms.csty = 0;
      cod->compparms.numdlvls = comp->numrlvls - 1;
      cod->prg = tile->prg;
       cod->numlyrs = tile->numlyrs;
      cod->compparms.cblkwidthval = JPC_COX_CBLKSIZEEXPN(comp->cblkwidthexpn);
      cod->compparms.cblkheightval=
              JPC_COX_CBLKSIZEEXPN(comp->cblkheightexpn);
      cod->compparms.cblksty = comp->cblksty;
      cod->compparms.qmfbid = comp->qmfbid;
      cod->mctrans = (tile->mctid != JPC_MCT_NONE);
      for (i = 0; i < comp->numrlvls; ++i) {
         cod->compparms.rlvls[i].parwidthval = comp->rlvls[i].prcwidthexpn;
         cod->compparms.rlvls[i].parheightval = comp->rlvls[i].prcheightexpn;
      }
      if (jpc_putms(encs[tileno]->tmpstream, encs[tileno]->cstate, encs[tileno]->mrk)) {
         ERROR();
         /* return -1; */
      }
      jpc_ms_destroy(encs[tileno]->mrk);
      encs[tileno]->mrk = 0;
       }
     }

for (cmptno = 0, comp = tile->tcmpts; cmptno < cp->numcmpts; ++cmptno, ++comp) {
   ccps = &cp->ccps[cmptno];
   if (ccps->numstepsizes == comp->numstepsizes) {
      samestepsizes = 1;
      for (bandno = 0; bandno < ccps->numstepsizes; ++bandno) {
         if (ccps->stepsizes[bandno] != comp->stepsizes[bandno]) {
            samestepsizes = 0;
            break;
         }
      }
   } else {
      samestepsizes = 0;
   }
   if (!samestepsizes) {
      if (!(encs[tileno]->mrk = jpc_ms_create(JPC_MS_QCC))) {
         ERROR();
```

```
            /* return -1; */
        }
        qcc = &encs[tileno]->mrk->parms.qcc;
        qcc->compno = cmptno;
        qcc->compparms.numguard = cp->tccp.numgbits;
        qcc->compparms.qntsty = (comp->qmfbid == JPC_COX_INS) ?
            JPC_QCX_SEQNT : JPC_QCX_NOQNT;
        qcc->compparms.numstepsizes = comp->numstepsizes;
        qcc->compparms.stepsizes = comp->stepsizes;
        if (jpc_putms(encs[tileno]->tmpstream, encs[tileno]->cstate, encs[tileno]->mrk)) {
            ERROR();
            /* return -1; */
        }
        qcc->compparms.stepsizes = 0;
        jpc_ms_destroy(encs[tileno]->mrk);
        encs[tileno]->mrk = 0;
    }
}
/* Write a SOD marker to indicate the end of the tile header. */
if (!(encs[tileno]->mrk = jpc_ms_create(JPC_MS_SOD))) {
    ERROR();
    /* return -1; */
}
if (jpc_putms(encs[tileno]->tmpstream, encs[tileno]->cstate, encs[tileno]->mrk)) {
    printf( "cannot write SOD marker¥n");
    ERROR();
    /* return -1; */
}
jpc_ms_destroy(encs[tileno]->mrk);
encs[tileno]->mrk = 0;
tilehdrlen = jas_stream_getrwcount(encs[tileno]->tmpstream);
if (jpc_enc_enccblks(encs[tileno])) {
    abort();
    ERROR();
    /* return -1; */
}
cp = encs[tileno]->cp;
rho = (double) (tile->brx - tile->tlx) * (tile->bry - tile->tly) /
    ((cp->refgrdwidth - cp->imgareatlx) * (cp->refgrdheight -cp->imgareatly));
     tile->rawsize = cp->rawsize * rho;

for (lyrno = 0; lyrno < tile->numlyrs - 1; ++lyrno) {
    tile->lyrsizes[lyrno] = tile->rawsize * jpc_fixtodbl(cp->tcp.ilyrrates[lyrno]);
}
tile->lyrsizes[tile->numlyrs - 1] = (cp->totalsize != UINT_FAST32_MAX) ?
    (rho * encs[tileno]->mainbodysize) : UINT_FAST32_MAX;
for (lyrno = 0; lyrno < tile->numlyrs; ++lyrno) {
    if (tile->lyrsizes[lyrno] != UINT_FAST32_MAX) {
        if (tilehdrlen <= tile->lyrsizes[lyrno]) {
            tile->lyrsizes[lyrno] -= tilehdrlen;
```

```
            } else {
                tile->lyrsizes[lyrno] = 0;
            }
          }
        }
        if (rateallocate(encs[tileno], tile->numlyrs, tile->lyrsizes)) {
          ERROR();
          /* return -1; */
        }
        if (jpc_enc_encodetiledata(encs[tileno])) {
          printf( "dotile failed¥n");
          ERROR();
          /* return -1; */
        }
        tilelen = jas_stream_tell(encs[tileno]->tmpstream);
        if (jas_stream_seek(encs[tileno]->tmpstream, 6, SEEK_SET) < 0) {
          ERROR();
          /* return -1; */
        }
        jpc_putuint32(encs[tileno]->tmpstream, tilelen);
        if (jas_stream_seek(encs[tileno]->tmpstream, 0, SEEK_SET) < 0) {
          ERROR();
          /* return -1; */
        }
        encs[tileno]->len += tilelen;

        jpc_enc_tile_destroy(encs[tileno]->curtile);
        encs[tileno]->curtile = 0;
    }/* endof #pragma omp parallel for */

    printf("%f second.¥n", second() - sec);

    /* put coded data to output stream in order */
    for(tileno = 0; tileno < numtiles; tileno++) {
        if (jpc_putdata(encs[tileno]->out, encs[tileno]->tmpstream, -1)) {
          ERROR();
          return -1;
        }
        jas_stream_close(encs[tileno]->tmpstream);
        encs[tileno]->tmpstream = 0;
        jas_free(encs[tileno]);
    }
    return 0;
}
```