卒業論文

PCクラスタを用いた
画像処理プログラムの並列化

氏　　名：粂山 直生
学籍番号：2210010074-0
指導教員：山崎 勝弘 教授
提 出 日：2005 年 2 月 21 日

立命館大学 理工学部 情報学科

内容梗概

高性能計算の手法の１つとして、１つの処理を複数台のコンピュータで同時に処理する並列計算が挙げられる。以前は並列処理というのは高価なスーパコンピュータのための技術であった。だが、ここ最近高性能なパーソナルコンピュータ(PC)やワークステーション(WS)が安価で入手できるようになったことからPCクラスタの技術が広まっている。PCクラスタとはPCなどの汎用のコンピュータを複数台ネットワークで繋ぐ事でひとつの並列コンピュータを実現したものである。

本研究ではOpenMPを用いて、「エッジ保存スムージングアルゴリズムによる画像の雑音除去プログラム」、「高次相関関数を用いたテクスチャ解析プログラム」の２つの画像処理プログラムの並列化を行った。OpenMP は、並列プログラムの中でも最近広く普及している共有メモリ環境用のプログラミング言語である。移植性が高く、段階的に並列化できるといったメリットを持っている。

エッジ保存スムージングは、雑音の混じった静止画像から雑音を除去するためのアルゴリズムである。静止画の雑音除去アルゴリズムには移動平均法、メディアンフィルタ法などがあるが、このアルゴリズムが最も除去能力に優れている。ただし、計算量もほかのアルゴリズムに比べて多い。

エッジ保存スムージングプログラムの速度向上比は 16 台で 10 倍程度であり、ある程度の並列化効果があった。

テクスチャ解析プログラムは、パターン認識のアルゴリズムである。Webからダウンロードした課題用のテクスチャ画像 2112 枚を用い、それぞれのテクスチャの特徴を抽出する。抽出した特徴を用いてテクスチャの識別を行い、高い識別精度を得た。

逐次では数時間かかる計算量の多いプログラムであり、処理のほぼ全体を並列化できたため、速度向上は16台で約16倍と理想的であった。

目次

1. はじめに 1
2. 並列処理とOpenMP 2
   2.1 PCクラスタ 2
   2.2 並列プログラミング 4
      2.2.1 OpenMP 4
      2.2.2 並列化による効果 4
3. エッジ保存スムージングの並列化 5
   3.1 問題定義 5
   3.2 アルゴリズム 5
   3.3 並列化手法 7
   3.4 実験 7
      3.4.1 実験条件 7
      3.4.2 実行結果 7
   3.5 考察 9
4. テクスチャ解析プログラム 10
   4.1 テクスチャ解析 10
      4.1.1 特徴抽出 10
      4.1.2 特徴選択 10
      4.1.3 テクスチャ識別 11
   4.2 マスクパターンによる特徴抽出 11
      4.2.1 マスクパターン 11
      4.2.2 マスクパターンの拡張(1)・付加特徴 12
      4.2.3 マスクパターンの拡張(2)・多様なサイズのマスク 13
      4.2.4 学習サンプル画像の特徴抽出 14
   4.3 特徴選択 15
      4.3.1 特徴選択の必要性 15
      4.3.2 特徴選択アルゴリズム 15
      4.3.3 クラス間分離度 J 16
      4.3.4 クラス内共分散・クラス間共分散 17
   4.4 テクスチャ識別 18
5. テクスチャ解析プログラムの並列化 20
   5.1 並列化手法 20
      5.1.1 特徴抽出プログラムの並列化 20
      5.1.2 テクスチャ識別プログラムの並列化 20

5.2 実験 21
5.2.1 実験条件 21
5.2.2 特徴抽出プログラム 21
5.2.3 テクスチャ識別プログラム 22
5.3 考察 24
5.3.1 特徴抽出プログラム 24
5.3.2 テクスチャ識別プログラム 24
6. おわりに 26
謝辞 27
参考文献 28

図目次

図 1：Goose 構成図 3
図 2：Raptor 構成図 3
図 3：局所領域 6
図 4：並列化手法 7
図 5：雑音除去結果 8
図 6：エッジ保存スムージング実行結果　単位：秒 9
図 7：テクスチャ画像の例 10
図 8：3x3 サイズの 224 個のマスクパターン 11
図 9：5x5 と 7x7 サイズのマスクにおける参照点 14
図 10：マスクパターンの例 14
図 11：クラス間分離度 J 17
図 12：2 つの分布とユークリッド距離 18
図 13：マハラノビス距離 19
図 14：特徴抽出プログラムの処理の流れ 20
図 15：テクスチャ識別プログラムの並列化 21
図 16：実行結果 22
図 17：テストサンプルの識別精度 23
図 18：実行結果 24

表目次

表 1：付加特徴算出式 12

1. はじめに

並列計算は大量のデータを用いて多くの計算を行う処理を高速化するための手法の１つである。ひとつの問題を解くのに何週間もの時間を要するのでは現実的ではない。処理時間を削減するために複数台のコンピュータで同時に処理を行うのが並列コンピューティングの考え方である。

現在、高速計算が求められるコンピュータの殆どは並列コンピュータである。有名な地球シミュレータ(NEC)や、その性能を 2004 年 9 月に超えた BlueGene/L(IBM)なども並列型である。

さらに近年になって上記のような大規模なコンピュータではなく PC などの汎用のコンピュータを高速ネットワークでつないで並列計算を行う PC クラスタが広まっている。PC クラスタはプロセッサの数に比例して実行速度が向上する、大規模な演算ができる、コストパフォーマンスが良い、といった長所を持っている[1]。

並列コンピュータには、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、プロセッサと主記憶から構成されたシステムが複数個互いに接続されていてプロセッサはほかのプロセッサの主記憶に読み書きをすることができない分散メモリモデル、分散モデルと同じように複数のシステムが互いに接続されているがプロセッサはほかのプロセッサの主記憶に互いに読み書きできる分散共有メモリモデルの 3 タイプがある。PC クラスタのそれぞれのノードは PC であるから、分散モデル、もしくは分散共有メモリモデルのタイプに属する並列コンピュータである。

並列計算の対象になるプログラムは、上記の通り一般に大きな値や多数の繰り返しを必要とする数値計算、流体計算、そして本論文で扱う画像処理などである。

また、本論文で作成したプログラムのうち、特にテクスチャ解析プログラムは最近のセキュリティに対する関心が高まるにつれて非常に重要視され始めている技術である。例えば、個人を認証する際に今まで ID とパスワードだけでチェックしていたところを本人の指紋をパスワードの代わりに用いる、というようなことである。

指紋に限らず、手のひらの静脈や顔、目の虹彩のような人の生体情報というものは個人を識別するに当たって非常に有用である。例えば全く同じ指紋を持つ人間はこの世に存在しない(現在の技術では 1 兆分の 1 程度の重複確率がある)。ほかの情報についても同様である。その情報が時間経過によって変化するようなものでなければ、それを個人認証に用いることはセキュリティの向上に貢献できる。本研究では、16 台の PC を Myrinet(Myrinet2000)と Ethernet(100Base-TX)で接続した Goose、8 台の PC を GigabitEther で接続した Raptor を用いて２つの画像処理プログラム「エッジ保存スムージングアルゴリズムによる画像の雑音除去プログラム」と「マスクパターンを用いたテクスチャ解析プログラム」を並列化し、考察する。PC クラスタシステムソフトウェアには SCore を採用している。またソフトウェア分散共有メモリシステム(SCASH)によって、分散メモリのクラスタ

上で共有メモリプログラミングができる。今回は共有メモリ型並列言語である OpenMP を使う。

本論文では、並列処理のディジタル画像処理への応用としてエッジ保存スムージングとテクスチャ解析を取り上げた。

エッジ保存スムージングは画像から雑音を除去するためのアルゴリズムである。元画像をあまり損ねることなく雑音を除去する、能力の高いアルゴリズムだがその反面処理量は多い。画素の輝度値と周囲の平均を取る処理がアルゴリズムの基本である。その処理を画素毎に OpenMP を用いて並列化した。

テクスチャ解析は、テクスチャ画像から特徴を抽出し、テクスチャ識別を行うためのものである。まず学習サンプルとして複数の画像の特徴を抽出しておく。この学習サンプルは多ければ多いほど識別精度を高めることができる。

次にテストサンプルを読み込み、学習サンプル群の特徴を用いてテストサンプルがどのような画像であるかを識別する。

本実験では学習サンプルとして 1200 枚を読み込み、テストサンプル 912 枚について識別を行った。テストサンプルごとの処理をノードに割り当てて並列実行した。

2. 並列処理と OpenMP

2.1 PC クラスタ

PC クラスタは並列コンピュータの 1 種である。以前は並列コンピュータには超高性能な処理を高速に実行するための高価で大規模なものしかなかった。現在もそうしたスーパコンピュータや大規模なサーバのようなマシンには並列型のものが多い。

だが、PC が急速に高性能で安価になってきたため、それらを複数台用いることで仮想的に 1 台の並列コンピュータとして利用することが広まってきた。これを PC クラスタという。PC クラスタは、2～8 台のごく小規模なものから数千台規模でスーパコンピュータとしての性能を発揮するものまで、予算や要求によって自由に構成することができるため非常にコストパフォーマンスに優れる。また上記の PC の高性能化とともに、それらを接続するネットワークも高速化が進み、コストパフォーマンスだけでなく、実際の性能で比較してもスーパコンピュータに引けをとらない高速計算の技術の 1 つであるといえる。

本実験では、Goose と Raptor という 2 台のクラスタを用いて実験を行なった。

クラスタ Goose の構成を図 1 に示す。

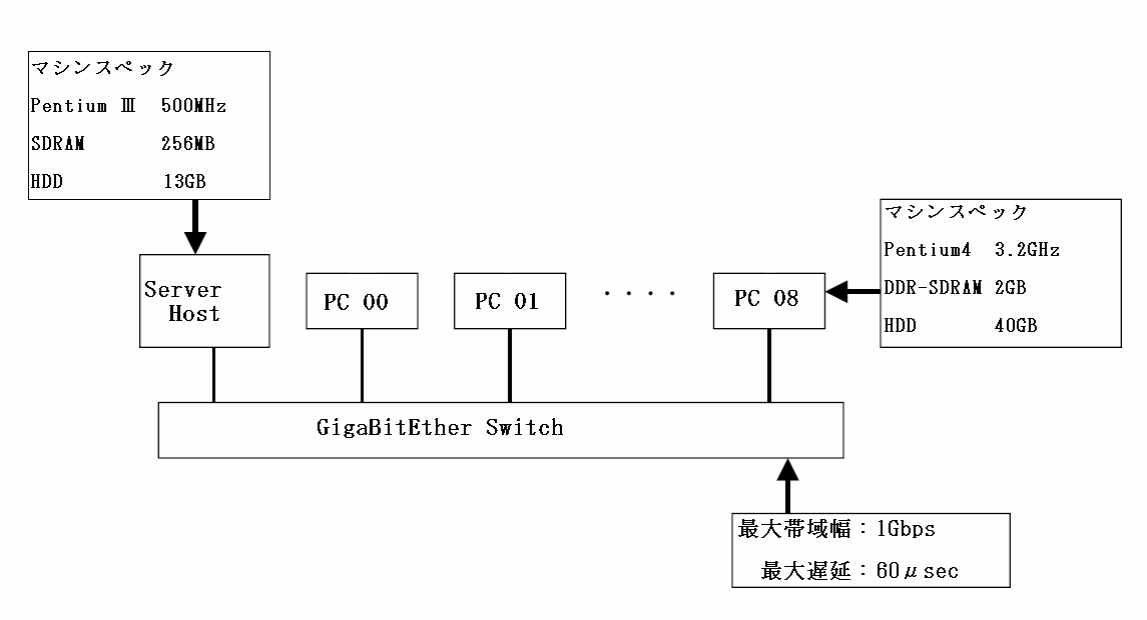


図 1：Goose 構成図

サーバホストと計算ノードは 100Base-TX でつながっている。更に、計算ノード同士を結合するのにはそれよりも更に早い Myrinet2000 を用いる。これはクラスタに主に使われる非常に高速なネットワークである。

クラスタ Raptor の構成を図 2 に示す。

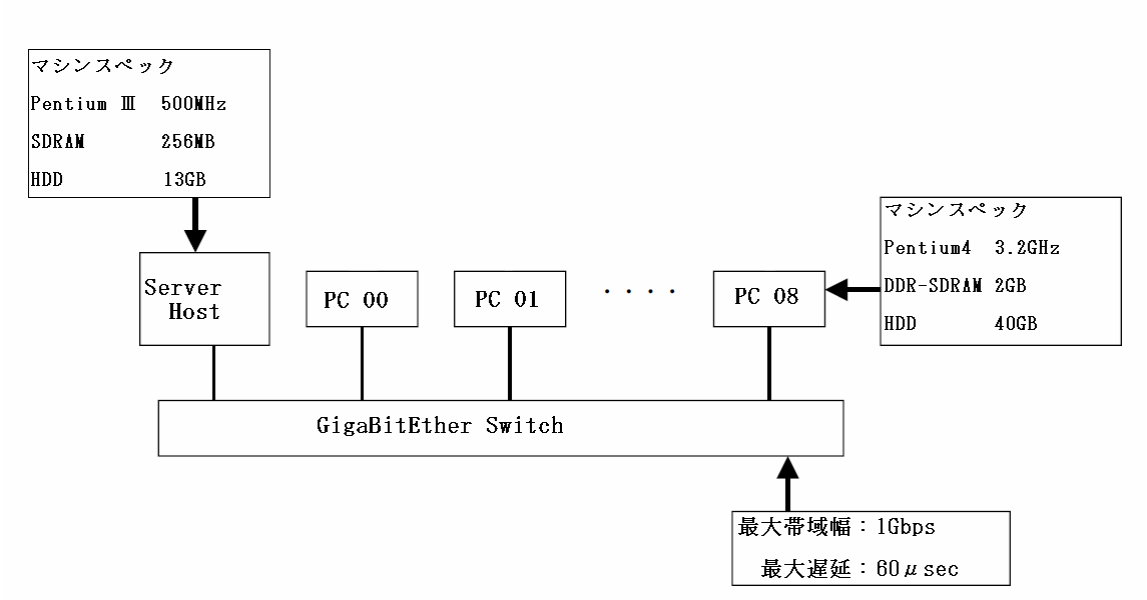


図 2：Raptor 構成図

計算ノード同士をつなぐためだけの高速ネットワークは使用しない。その代わり、サー

バと全てのノードを高速なGigaBitEtherで結合する。

2.2 並列プログラミング

2.2.1 OpenMP

OpenMP は、共有メモリ並列プログラミングのための言語である。1997 年に OpenMp Architeture Review Board によって Fortran をベースとして開発された API 使用である。その後 C/C++用にも API 仕様が開発された。

OpenMP は Fortran や C/C++をベースにコンパイラ指示文、ライブラリ、環境変数によって並列処理を行えるように拡張したものである。したがって、プログラムのソースコードの中に逐次部分と並列部分が存在する。

OpenMP が実行されるとマスタスレッドという単一のスレッドのみで実行が開始される。並列化指示文を読むまでは（つまり、逐次処理部分では）処理を行うスレッドはマスタのみである。プログラムの実行が並列指示文に達するとスレーブスレッドとよばれる複数のスレッドを生成し、並列指示文で指定されている範囲の処理が全スレッドにおいて実行される。この部分をパラレルリージョンと呼ぶ。

2.2.2 並列化による効果

並列に処理を実行するということは、用いるスレッドの数だけ実行速度の向上が期待できるように感じられる。つまり、2 台で並列実行すれば実行時間は 1/2 に、4 台では 1/4 に、といった具合である。

しかし、実際にプログラムを並列で実行してもそのような（期待したほどの）速度向上は得られないことが多い。その理由として、以下の 3 つが挙げられる。

- 逐次処理部分のオーバヘッド

  並列効果を推測するための方法として、アムダールの法則と呼ばれる計算式がある。それによると、プログラムの中で逐次処理部分の割合が全体の $s(0 \le s \le 1)$ 存在すれば n 台のプロセッサを用いて得られる速度向上は

  $$\frac{1}{s+\frac{1-s}{n}} \quad \text{----------} (1)$$

  となる。この式に従えば、プログラムの中に逐次実行部分の割合が 5%あった時に 10000 台のプロセッサを用いて並列実行して得られる速度向上は高々20 倍程度にしかならない。

  このことは、アルゴリズムを工夫するなどして、できる限り並列部分の割合を大きくすることが並列計算の効率を上げるために非常に重要であることを示している。

- 負荷均衡

  並列処理で各プロセッサに処理を割り当てるときに、それぞれのプロセッサが受け持つ処理の負荷は当然均一にはならない。したがって、全体の実行時間はもっとも実行時間の遅いプロセッサで決まる。並列化効果を上げるためには、処理の負荷均衡をうまくとる分散方法が重要である。

- 通信/同期のオーバヘッド

  複数台のマシンで協力し合って 1 つの処理をこなすのだから、当然ネットワークを介してデータや処理要求のやり取りを行ったり、同期を取ったりという作業が必要である。その通信にかかる時間が全体の実行時間のボトルネックになる。これを回避するためには、高速なネットワークを採用したりデータのやり取りの回数をできる限り減らすといった対策を講じなければならない。

  ただし、今回の実験は共有メモリ環境で行なう。共有メモリ環境においてはプログラム中でデータ通信を行なう必要がないので、考慮すべきなのは同期のオーバヘッドだけである。

3. エッジ保存スムージングの並列化

3.1 問題定義

エッジ保存スムージングは、インパルス雑音の入った画像から雑音を除去するためのアルゴリズムの 1 つである。インパルス雑音とは、画像の不規則な位置に不規則な大きさで発生する雑音のことである。これに対して、ガウス分布に沿って規則的に発生した、雑音がどこにどんな風に発生しているか把握できる雑音をガウス雑音と呼ぶ。

インパルス雑音を除去するためのアルゴリズムには、移動平均法、メディアンフィルタ法などがある。エッジ保存スムージングはそれらのアルゴリズムの中で最も除去能力に優れるものである。ただし、計算量も多くなる。

3.2 アルゴリズム

静止画像の雑音除去は、動画のそれよりも精度が落ちる。なぜなら、動画は雑音によって失われた画像の情報を時間的に近い位置にある部分、例えば 1 コマ前の画像などから推測できるのに対し、静止画ではそれができないからである。

静止画における雑音処理で、雑音によって失われた画像情報を類推するには空間的に近い位置にある部分を用いるしかない。つまり、雑音の周囲の画素である。一般に、画像中のある画素とその隣の画素値は殆ど等しいことが多い。しかし、もしある画素に雑音が発生していたとすると、その隣の画素の画素値とはまったく違う画素値になるはずである。

そこで注目画素とその周囲の画素との平均をとることで、画像に含まれる大きく画素値が変化する部分をぼかし、目立たなくする。この処理を平滑化といい、静止画の雑音を除去するための基本的な方法である。

しかし、大きく画素値が変化するのは何も雑音のある部分だけではない。画像にはもともと大きく画素値の変化する部分が含まれている。これを画像のエッジ部分と呼ぶ。上記の方法では、雑音だけでなく元の画像が持つエッジ部分までぼかしてしまい、全体的にぼやけたような画像になってしまう。

それを避け、雑音だけを除去するアルゴリズムがいくつか考案されている。エッジ保存スムージングもそのうちの 1 つである。

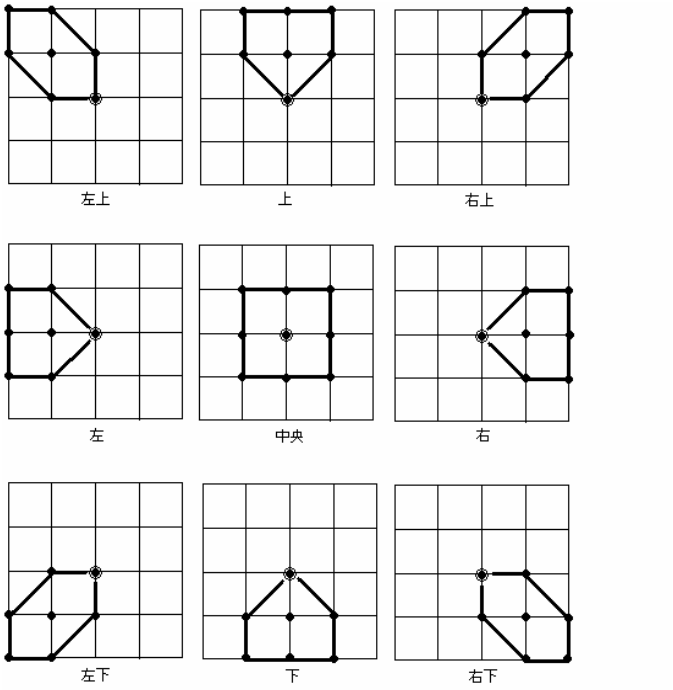


図 3：局所領域

まず注目画素の周囲 5x5 の画素からなる領域を考える。その領域の中から、さらに細かな 9 つの局所領域を考える(図 3)。これらの局所領域は 7 つ若しくは 9 つの画素から構成されている。それぞれの局所領域内の分散を計算した時、もしその領域内にエッジが存在すればその領域における分散は大きくなるはずである。したがって、9 つある局所領域の中で最も分散の小さいものは、エッジを含まない局所領域であるといえる[8]。

元の画像をぼかすことなく雑音を除去するためには、最も分散の小さい領域を見つけ、その領域内の平均濃度を画素値として採用すればよい。このように注目画素の周囲の領域の平均を取る際に何らかのパラメータを基準にする方法は選択的局所平均化と呼ばれる。エッジ保存スムージングはそのうちの 1 つである。

この操作をすべての画素において実行することで雑音を除去するのがエッジ保存スムー

ジングアルゴリズムである。

3.3 並列化手法

画像データは、画像の画素数と同じ大きさの 2 次元配列で表現する。元画像と処理済み画像を格納する配列をそれぞれ用意する。それぞれの画素に対しての処理は依存性を持たず、完全に独立している。したがって、処理する画素を各ノードに均等に割り付けることでほぼすべての処理を並列化することが可能である(図 4)。

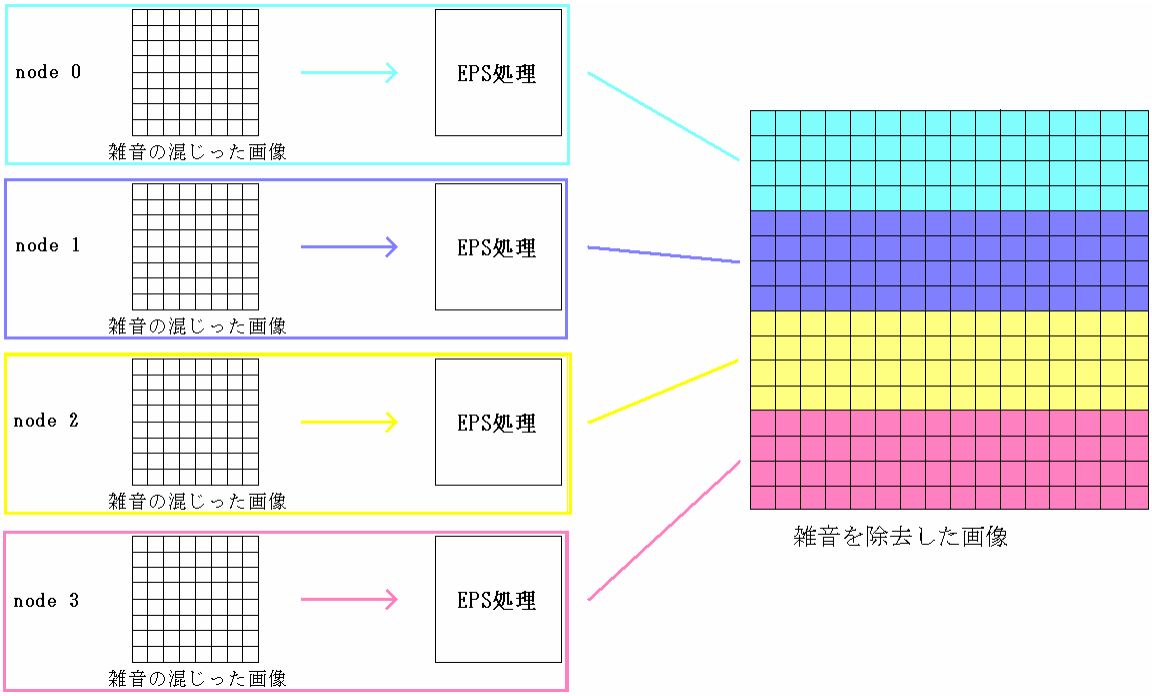


図 4：並列化手法

3.4 実験

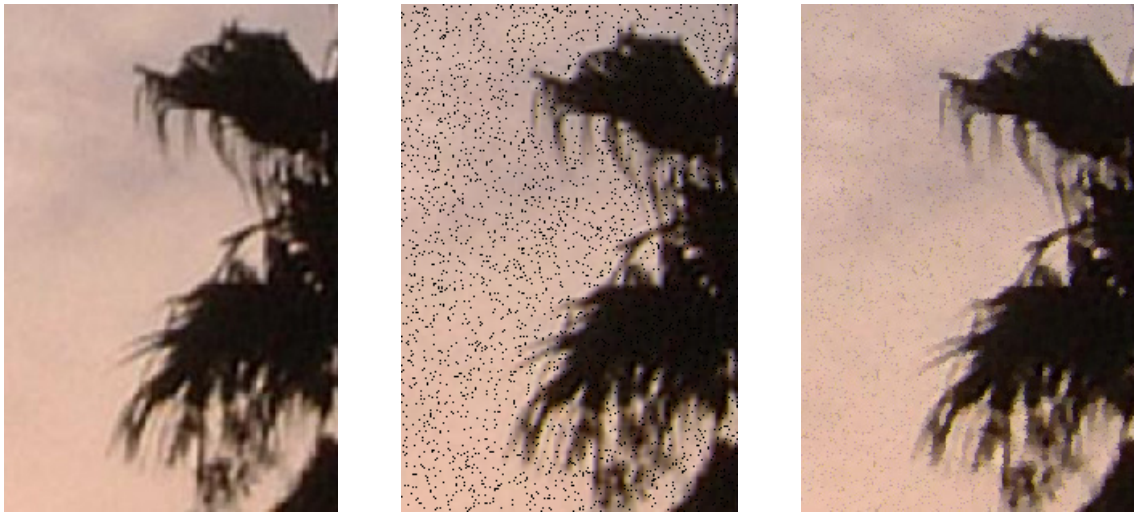
3.4.1 実験条件

本実験では、2304x1728 サイズのカラー画像に雑音を発生させたものを処理の対象として用いた。発生させた雑音はインパルス雑音と呼ばれるものである。インパルス雑音とは、雑音の発生位置や大きさに規則性が見られない雑音のことをいう。元の画像のそれぞれの画素をランダムに約 3%の確率で画素値 0 にする(真っ黒な画素にする)ことで雑音を発生させる。実験には論文を使用し、1、2、4、8、16 台でそれぞれ実行した。

3.4.2 実行結果

元画像、雑音を発生させた画像、雑音除去処理後の画像の 1 部を図 5 に示す。かなりの

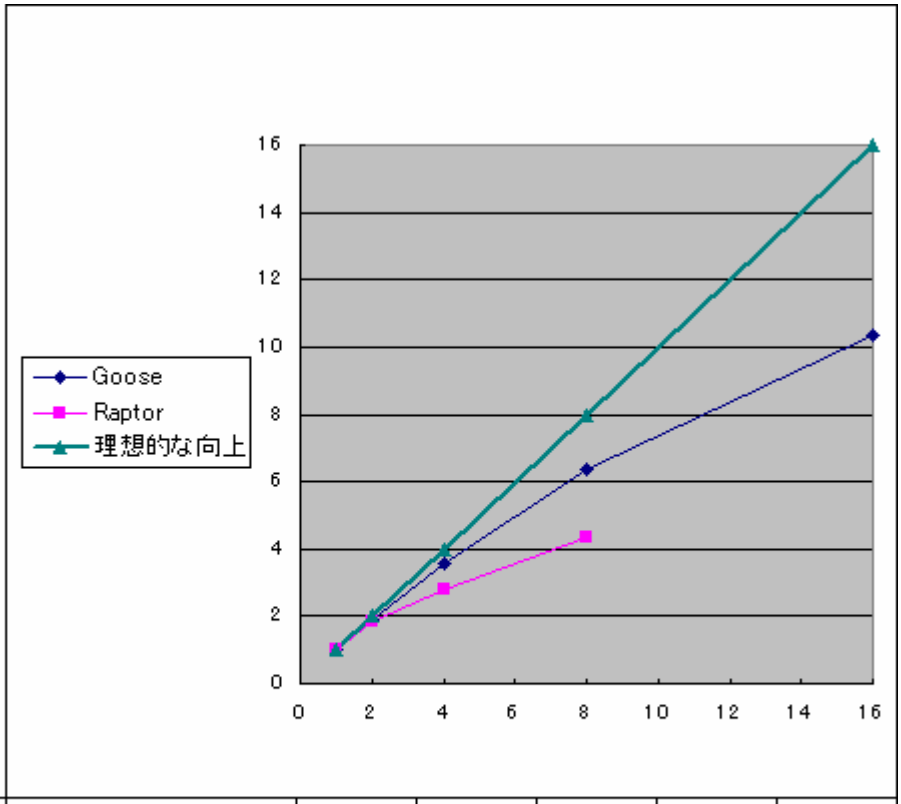
量の雑音が入っているが、(a)と(c)を見比べてみるとそれほど画像を劣化させずに取り除けている。



(a)元画像　　(b)雑音を含んだ画像　　(c)除去後の画像

図 5：雑音除去結果

Goose と Raptor で実行し、実行時間を計測した結果を図 6 に示す。16 台で 10 倍強の速度向上が得られた。



Goose	実行時間	136.44	70.93	38.04	21.49	13.16
	速度向上比	1	1.92	3.58	6.35	10.36
Raptor	実行時間	24.06	13.12	8.86	5.2	-
	速度向上比	1	1.83	2.78	4.36	-

図 6：エッジ保存スムージング実行結果　　単位：秒

3.5 考察

このプログラムの中で並列化している部分は、雑音を除去するための計算を実際に行っているモジュールだけである。その他の部分は逐次で処理を行う以外に方法がなかった。逐次処理部分の中でも時間がかかり速度向上のボトルネックになっているのはファイル入出力モジュールである。

この部分はある程度の処理の大きさを持っている上に、ネットワークを用いて通信を行なわなければならないので速度向上の妨げになっていると考えられる。Goose に比べて Raptor の速度向上が低いのは、全体に対する通信の量が影響している。Raptor クラスタの計算ノードの CPU は 6 倍以上のスペックを持っている。しかし、Raptor に用いられているネットワーク GigaBitEther は Goose の Myrinet2000 に比べてかなり遅い。そのため Raptor は通信のネットワーク速度に処理速度が引っ張られて遅くなってしまった。

また、雑音除去処理を行なっているモジュールも全てを並列化できているわけではない。画像の画素数がノードの台数で割り切れなければ、その剰余の部分は逐次処理するしかなく性能のボトルネックとなる。台数が増えるにつれて速度向上比が下がっているのは、剰

余の部分が増えているからだと考えられる。

4. テクスチャ解析プログラム

4.1 テクスチャ解析

テクスチャ識別とは、パターン認識の技術を用いてテクスチャ画像から特徴を抽出し、学習サンプルを基にしてテストサンプルを識別するものである。以下にサンプルとクラスの例を図 7 に示す［11］。ここに示したようなテクスチャ画像を用い、クラスを識別する。

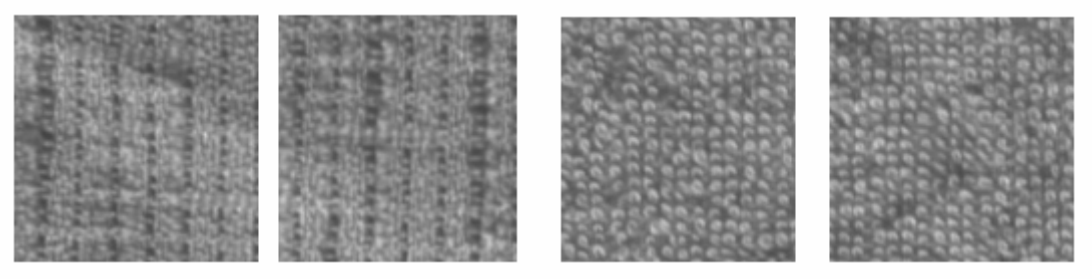


図 7：テクスチャ画像の例

クラスとは、同じような特徴を持った画像の集合である。図 7 の画像は全て違う画像であるが、左 2 つ、右 2 つはかなり似ているのがわかる。例えばある画像をテストサンプルとして読み込んだとき、左側のクラスに属するのか、右側のクラスに属するのかを識別するのがこのプログラムの目的である。

下に示す 3 つの処理をそれぞれ独立したプログラムとして作成する。なお、特徴選択プログラムは依存性が高いため、並列化は行なわない。C プログラムとして、Raptor の計算ホストの上で実行すると、実行に約 30 分かかった。

4.1.1 特徴抽出

まず最初にすべきことは、学習サンプル画像群からそれぞれ特徴を抽出し、クラスを定義することである。このとき読み込み、算出したそれぞれのサンプルの特徴ベクトルの分布がクラスをあらわすパラメータになる。このとき読み込む学習サンプルの数は多ければ多いほど識別精度を高めることができるが、多すぎると処理に膨大な時間を要してしまう。

4.1.2 特徴選択

抽出された複数の特徴をそのまま使うことはできない。特徴ベクトルの次数が、読み込んだ学習サンプルに対してあまりに多いと、識別するときに計算量が膨大になるだけでなく識別率が下がってしまうことがあるからである。この詳細については 4.3.1 項で述べる。これを避けるために、702 個の特徴の中からそれぞれのクラスを最もよく識別する特徴を選

択する必要がある。そのために特徴選択プログラムを作成する。

4.1.3　テクスチャ識別

特徴抽出、特徴選択プログラムによって学習サンプルから特徴ベクトルが生成された。次に、識別したいテクスチャ画像を読み込む。これをテストサンプルと呼ぶ。このテストサンプルの特徴ベクトルと、学習サンプルによって構成されているクラスとその特徴ベクトルを用いて、テストサンプルがどのクラスに属しているのかを識別する。

識別に用いるのはマハラノビス距離(Mahalanobisdistance)という量である。マハラノビス距離とは、共分散行列の同じ分布の平均距離をあらわす量である。今回は、読み込んだテストサンプルの 1 点と、それぞれのクラス(分布)の間の距離をあらわす様にマハラノビス距離を変更して用いた。

テストサンプルの特徴ベクトルを座標とする。テストサンプル、各学習サンプルを、特徴ベクトルの次元(特徴選択を行なわなければ 702 次元)の点として表現する。クラスは、そのクラスに属する各点(各学習サンプル)の分布という形であらわされる。

4.2　マスクパターンによる特徴抽出

4.2.1　マスクパターン

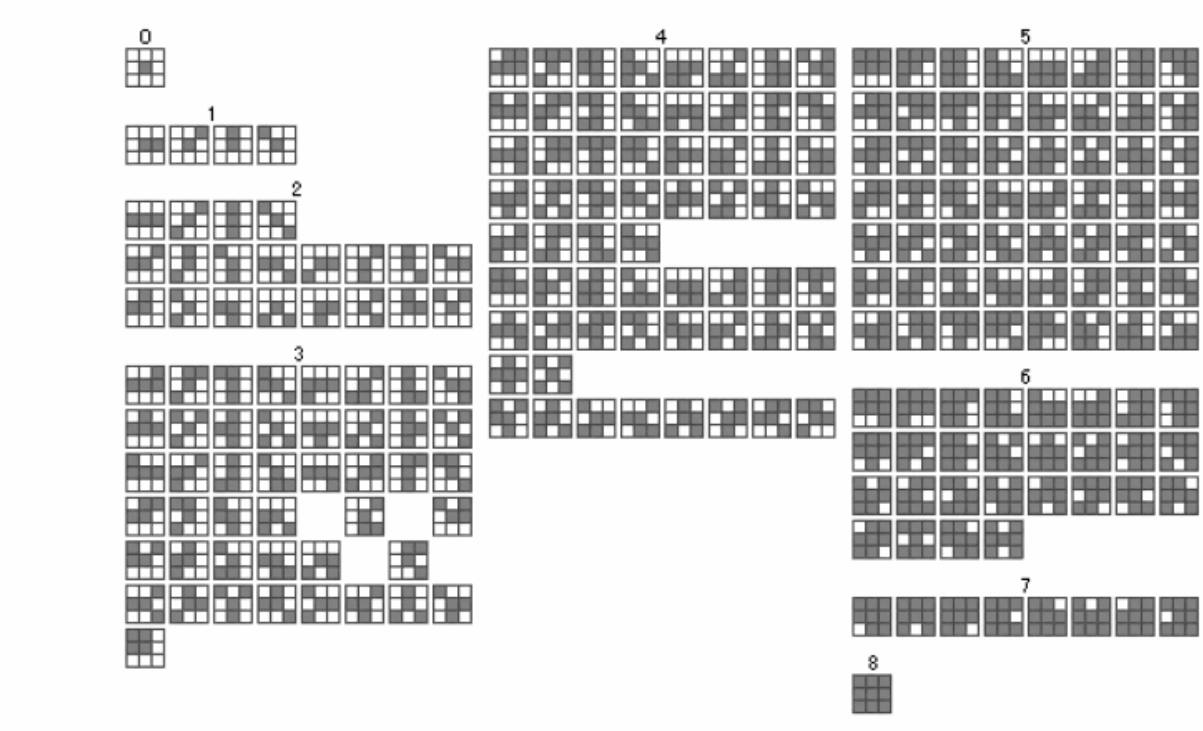


図 8：3x3 サイズの 224 個のマスクパターン

サイズ、カラーなのかグレースケールなのか、平均画素値など画像を現す特徴はいろい

ろと考えられる。本実験で用いた特徴は高次局所自己相関特徴と呼ばれるものである。多様なサイズ、種類のマスクパターンを用いることで、画像の各画素における輝度値の分布を画像の特徴として用いるものである。画像の周囲 8 画素という局所領域において、参照する画素の数を次数と呼ぶことにする。マスクパターンによっては最大で 9 次の計算を行なうことから、高次局所特徴と呼ぶ。

図 8 に示した 224 個のマスクパターンは、画像の周波数成分を特徴として抽出するものである。3x3 領域の周囲 8 点を参照するのであるから 2 の 8 乗で 256 通りが考えられるが、後で全ての画素の値を足し合わせるため、水平方向に移動した場合等価となるパターンを除き 224 通りとしている。

図 8 において、真ん中の画素を注目画素とする。図のパターンにおいて、網掛けになっている部分の画素をすべて掛け合わせた値をそのパターンのその画素における特徴とする。例えば 1 番左上のマスクでは注目画素のみが選ばれているので、特徴量は注目画素の値と同じである。その下のマスクでは、特徴は注目画素とその右隣の画素を掛け合わせた値になる。

但し掛け合わせた数によって特徴の大きさが変わってくるので、その偏りを消すために画素値を平均値で割る。更に n 回掛けた場合は算出した特徴の n 乗根を求め、それを特徴量として用いることにする。この処理を行なうことで、各特徴の平均値は約 1.0 になる。これを特徴の正規化という [9]。

このようにして 1 つの画素につき 224 通りの特徴が算出される。すべての画素のそれぞれの特徴を足し合わせ、1 つのテクスチャ画像から 224 個の特徴を抽出する。

4.2.2 マスクパターンの拡張(1)・付加特徴

前節では 224 個の 3x3 サイズのマスクパターンについて説明した。しかし、より多様な種類の特徴を抽出するため、前節で提案した特徴をベースにさらに様々なマスクパターンを考える。

文献[12]では、参照演算（輝度値の積演算）を、注目画素も含めマスクパターンの同一箇所に対して行なっている。その算出式を表 1に示す。

表 1：付加特徴算出式

	マスクパターンの周囲参照点数 0個	マスクパターンの周囲参照点数 1個
	f(r)	f(r)xf(r+a1)
付加特徴（変位）	f(r)xf(r) (a1=0)	f(r)xf(r+a1)xf(r) (a2=0)
付加特徴（変位）	f(r)xf(r)xf(r) (a1=a2=0)	f(r)xf(r+a1)xf(r+a1) (a1=a2)

ここで、表の $f(r)$ は中心着目点の輝度値、$f(r+a_i)$ は周囲参照点の輝度値を表している。また a_1 や a_2 はマスクパターンを表現するのに用いる。マスクパターンによって掛け合わされる輝度値の個数も変わる。例えば、注目画素を含め 4 つの画素値を掛け合わせるマスクパターンを考える。その場合、マスクパターンに基づく計算は $a_0 \times a_1 \times a_2 \times a_3$ と表現する。最大で 9 個の画素値を掛け合わせるので、a_0～a_8 までの変位が存在する。

a_i はそれぞれ 0～8 の値を持つ。この値は注目画素を中心とした画素の位置を表すものである。0 は注目画素、1 は注目画素の左上の画素、2 は注目画素の真上、というように数字を注目画素の周囲の画素の位置情報とする。a_0～a_8 に 0～8 の値を割り振ることでマスクパターンを表現する。例として注目画素、その左上、左横、増した、の 4 つの画素値を掛け合わせるマスクパターンについて説明する。$a_0=0$、$a_1=1$、$a_2=4$、$a_3=7$、と表現することになる。また、a_0 は常に注目画素を表している。

表において、例えば $a_1=0$ と書いてある場合、$a_0=0$、$a_1=0$ というマスクと同じ意味になる。つまり、付加特徴とは同じ画素を重複して書けることで生成する特徴である。

全てのマスクパターンに対して付加特徴を設けると、特徴の数が膨大になってしまう。そこで、マスクのうち周囲参照点が 0 もしくは 1 であるものに対してのみ付加特徴を設定する。周囲参照点が 0 のパターンは当然 1 つ、そして参照点が 1 のパターンは(水平方向に等価なものを除くので 8 種類ではなく)4 種類である。合計 5 種類のマスクパターンに対してそれぞれ 2 つの特徴を付加特徴として選ぶ。合計で、10 個の付加特徴を上記の 224 個の特徴に加え、234 個のマスクパターンを設定する。

4.2.3 マスクパターンの拡張(2)・多様なサイズのマスク

今までに設定したマスクパターンは全て3x3領域の周囲8近傍だけを考えたものである。しかし、3x3のサイズだけでは画像の高周波成分は抽出できるが低周波成分は抽出できない。そこで、5x5、7x7サイズのマスクパターンも用意することにする。

しかし、5x5で周囲25点、7x7で周囲49点もの画素を参照していたのでは特徴数が膨大になってしまう。そこで、新たな2つのサイズのマスクは周囲25、49点からそれぞれ8点のみを参照することとする。そして3x3サイズのマスクと1対1で対応できるように設定する。これによって計算量の増大を回避する。

説明してきたマスクパターンの設定方法に則って、それぞれ 234 個、全てあわせて 702 個のマスクを考えることにする(図 9)。

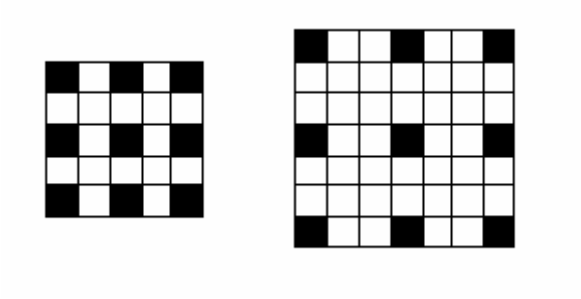


図 9：5x5 と 7x7 サイズのマスクにおける参照点

図 9 において黒くなっている部分の画素が 3x3 サイズのマスクの参照点とそれぞれ対応している。3x3 サイズで注目画素とその右隣の画素を掛け合わせるマスクパターンにおいて、5x5 サイズでは注目画素とその 2 つ右隣、7x7 サイズではその 3 つ右隣の画素をそれぞれ掛け合わせて特徴とする。

4.2.4　学習サンプル画像の特徴抽出

学習サンプルとして複数の画像を読み込み、上で述べた 702 個のマスクパターンを用いて 1 枚の画像から 702 個の特徴を抽出する。抽出した特徴を 1 つの 702 次元特徴ベクトルとして配列に格納する。その例を図 10 に示す。

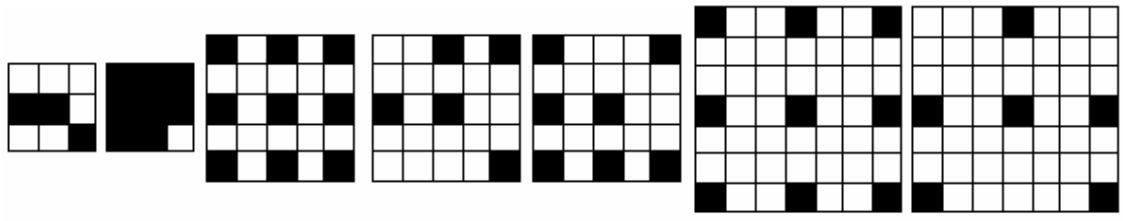


図 10：マスクパターンの例

図 10 に示したマスクパターンからそのパターンにおける特徴量を実際に算出する方法を、左端のマスクパターンと右端のマスクパターンを例にとって説明する。

左端のマスクパターンの場合、黒くなっているのは注目画素と、その左隣の画素、右下の画素である。それらを掛け合わせる。そうやって算出された量の 3 乗根をとり、この値をこのパターンのその画素における特徴量とする。右端のパターンでは、注目画素と、その 3 つ上の画素、3 つ左の画素、3 つ右の画素、3 つ左下の画素、3 つ右下の画素の 6 つを掛け合わせ、その 6 乗根をとる。

これらの計算を画像の全ての画素に対して行ない、それらを足し合わせることで、1 つの画像からこのパターンに対応する 1 つの特徴量を算出する。

ただし、7x7 サイズのマスクでは少なくとも注目画素の外側に 3 つの画素が存在しないと

特徴量の算出を行なうことができない。そこで、特徴量の算出は画像の外側３周分を除いた画素に対してのみ行なう。

4.3 特徴選択

4.3.1 特徴選択の必要性

テクスチャ識別の精度を上げる方法として考えられるのは、特徴の種類と学習サンプルとして読み込むテクスチャ画像の数を増やすことである。しかし、多くの学習サンプルを読み込むのは非常に時間がかかる。

実行に余りに時間がかかりすぎるようではパターン認識プログラムを実用することはとてもできない。パワーのあるコンピュータを用いれば実行時間を短縮することはできるが、それではコストがかかってしまう。

では、特徴の種類を増やせば識別率を上げることができるように思われるかもしれないが、それ程簡単にはいかない。学習サンプル数と違って、特徴の種類を増やす、という作業は慎重に行なわなければ逆効果になることがあるのである。

学習パターンの数を n とする。特徴数 d が n に比べてはるかに小さい場合、新たに特徴を増やすと識別精度を上げることができる。

ところが、d が n と比べて無視できないほど大きくなってくると統計的な信頼が低下し、逆に識別率の低下を招いてしまうことがある。これをヒューズの法則と呼ぶ。

今回読み込んだ学習サンプル画像の数は 1200 枚である。特徴数に対して十分多いとはいえない。このままでは、前述のヒューズの法則にしたがって識別率が悪くなってしまう可能性が高く、また計算量も膨大になってしまう。そこで、702 個から特徴を削減する必要がある。しかし、ただ何の指針も無く特徴を選んだだけでは高い識別率は望めない。そこで、最も各画像や各クラスの分布を識別するのに適した特徴を選択する必要がある。

4.3.2 特徴選択アルゴリズム

従来から利用されている基本的な特徴選択手法を採用する。学習サンプル画像のみを用いて 702 個の特徴から実際に識別に使用する特徴を選択するアルゴリズムを以下に示す。選択する特徴の量によって識別率も変わってくるが、最大でも 49 個以下にとどめておかなければならない。その理由は、各クラスの学習サンプルが最大 50 枚ずつしかないためである。今回用いた識別アルゴリズムでは、特徴ベクトルの次元数が学習サンプル以上になると識別が完全にできなくなってしまう(4.4 節で詳述)。これは前項で述べたヒューズの法則とはまた別の理由による。

702 個の特徴は選択候補とし、原特徴集合と呼ぶことにする。

STEP1：各学習サンプル画像(1200 枚)から、上記の原特徴集合を抽出し、配列選択済み特徴集合を初期化する。

STEP2：原特徴集合のうち未選択の特徴すべてについてそれぞれを既存の選択済み特徴集合に加えて特徴を構成したときの学習サンプルに対するクラス間分離度 J を計算する。（次節で詳述）

STEP3：前ステップで算出されるクラス間分離度が最大になる特徴を新たに選択済み特徴集合に加える。

STEP4：設定した数の特徴を選択し終えるまで STEP2 と STEP3 を繰り返す。

4.3.3　クラス間分離度 J

各クラスがどれだけ離れているかを示す量として用いる。クラス間分離度（以後 J ）の計算法は以下のとおりである。

$$J = tr\left(\sum_W^{-1}\sum_B\right) \qquad \text{----------} \ (2)$$

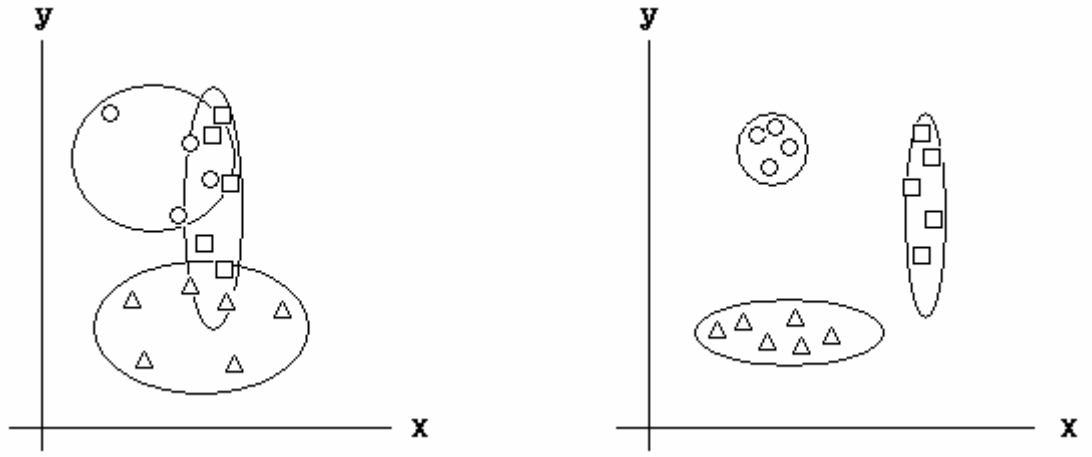
Σ_W ：クラス内共分散行列

Σ_B ：クラス間共分散行列

ここで、クラス内共分散はクラス間の学習サンプルのばらつき、クラス間共分散はそれぞれのクラスのばらつきを示し、選択した特徴数を d とすると、それぞれ $d \times d$ 次元の行列である。また、J が大きいほうが識別精度が上がる。

図 11はクラス間分離度が識別に与える影響について示したものである。x と y はサンプルの特徴量である。つまり、特徴が2つである場合の模式図である。特徴ベクトルが2次元であるとすると、1つのサンプルは2次元座標系上の1つの点として表現することができる。

図 11とクラス内共分散、クラス間共分散を説明する。クラス内共分散は3つの楕円それぞれの大きさであり、3つの楕円がどれだけ離れているかを示す。式(2)に示すように、J はクラス内共分散行列の逆行列とクラス間共分散行列を掛けた行列の対角成分である。



(a)クラス間分離度 小　　(b)クラス間分離度 大

図 11：クラス間分離度 J

図 11 が示すように、J が大きければ大きいほど識別の精度は上がる。したがって、J がなるべく大きくなるような特徴の組み合わせを選択することが識別率を上げるために重要となる。

4.3.4　クラス内共分散・クラス間共分散

クラス内共分散行列Σ_iは、式(3)で表される正則行列である。

$$\sum_w \overset{def}{=} \sum_{i=0}^{24} p(\omega_i)\Sigma_i \qquad \text{----------} \qquad (3)$$

ここで、$p(\omega_i)$はクラスの生起確率をあらわす。またΣ_iはそのクラスの分布の様子を表す共分散行列であり、式(4)のようになる。

$$\sum_i \overset{def}{=} \frac{1}{n}\sum_{i=0}^{24}(x-m_i)(x-m_i)^t \qquad \text{----------} \quad (4)$$

nはクラス内の学習サンプルの数、xはサンプルの特徴量、m_iはクラス平均である。したがって、例えば特徴がd個の場合は$(x-m_i)$はd次元ベクトルであり、クラス共分散、クラス内共分散共に$d \times d$の行列になる。

また、クラス間共分散行列は式(5)である。

$$\sum_B \overset{def}{=} \sum_{i=0}^{24} p(\omega_i)(m_i-m)(m_i-m)^t \qquad \text{-----} \quad (5)$$

クラス間共分散行列もまた $d \times d$ 行列である。この行列はそれぞれのクラスの分布の様子を示す。ここで m は、すべてのクラスの平均である。

4.4 テクスチャ識別

識別は、識別したい画像と各クラスを比較して最もその画像に近いクラスを見つける作業である。識別したい画像をテストサンプルと呼ぶ。

まず、テストサンプルを読み込み、読み込んだ画像データからテストサンプルの特徴を抽出する。このアルゴリズムは 4.2 節で学習サンプル群の特徴を抽出するときに用いたものを用いる。ただし抽出する特徴は特徴選択によって算出された d 個の特徴のみとする。

テストサンプルを d 次元座標系上の 1 点とする。すると、24 のクラスは、それぞれに属する学習サンプル群の点の分布で表されることになる。24 の分布とテストサンプルの点の距離を求める。その距離が最も小さいクラス、つまりテストサンプルとの距離が最も近いクラスをテストサンプルが属するクラスであると判断する。

しかし、単純なクラスの平均(重心)とのユークリッド距離を用いて識別を行っても期待通りの精度は出ない。その理由は、ユークリッド距離が分布の分散を考慮に入れていない距離であるからである。

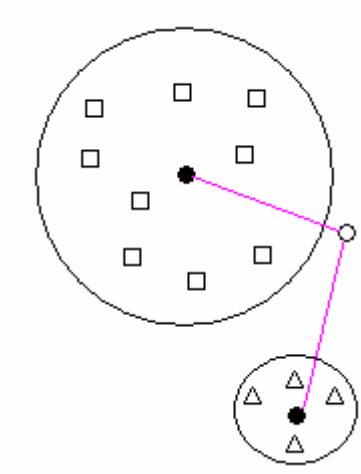


図 12：2 つの分布とユークリッド距離

図 12 で、赤線は○と●とのユークリッド距離、□と△はそれぞれ分布に属する点、●は分布の重心とし、○の点(テストサンプル)がどちらの分布に近いかを判断したいものとする。2 つの分布は、ユークリッド距離だけを考慮すればテストサンプルから同じ距離にある。

しかし、2 つの分布の分散は大きく違う。左上の分布は右下のものよりも分散が大きく、広範囲に広がっている。分布の重心だけでなく、その広がりも考慮するとテストサンプルと近い距離にあるのは左上の分布であるといえる。しかし、ユークリッド距離では比較す

る分布の広がりを表現することができないため、分布同士や分布と点の間の距離をあらわすのには向いていない。そこで、本実験では分布の分散を考慮した距離であるマハラノビス距離を識別に用いる。

マハラノビス距離はマハラノビス汎距離とも呼ばれ、共分散の等しい 2 つの分布間の距離を表すためのものである。今回はそれを 1 点と 1 つの分布との距離をあらわす単位に拡張して用いた。

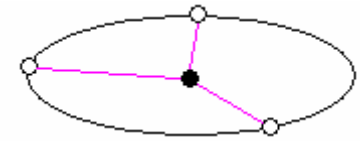


図 13：マハラノビス距離

今、ある分布が図 13 のように楕円形をしているとする。●が分布の重心、赤線が 3 つの点と分布の重心とのユークリッド距離をあらわしている。3 本の線はそれぞれ長さが異なる。したがって、3 つの点は分布からのユークリッド距離は違っている。しかし、分布の形が楕円形になっているため、3 点と分布とのマハラノビス距離は等しいということがいえる。

今、クラス i の平均ベクトル m_i および共分散行列 Σ_i が

$$m_i = \begin{pmatrix} m_1 \\ \vdots \\ m_n \end{pmatrix}、\quad \Sigma_i = \begin{pmatrix} \Sigma_{11} \cdots \Sigma_{1n} \\ \vdots \quad \ddots \vdots \\ \Sigma_{n1} \ldots \Sigma_{nn} \end{pmatrix}$$

である時、

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

とクラス i のマハラノビス距離は

$$tr(x-m))\Sigma_i(x-m) \quad \text{----------(6)}$$

で表される。テストサンプルと全てのクラスに対してこの計算式を用いてマハラノビス距

離を計算し、最もマハラノビス距離が小さかったクラスがテストサンプルの属しているクラスであるという判断をする。

5. テクスチャ解析プログラムの並列化

5.1 並列化手法

5.1.1 特徴抽出プログラムの並列化

複数の学習サンプル画像の特徴を抽出するプログラムである。学習サンプルが多ければ多いほど識別精度は高まるので、大量の画像を学習サンプルとして読み込み、処理する必要がある。

1 枚の画像を読み込み、4.2 節のアルゴリズムを用いて 1 つの特徴ベクトルを算出する処理(図 14)はそれだけで完結している。したがって、他の部分の処理が影響したり、逆に影響を与えたりすることはない。そこで、それぞれのノードに画像単位の処理を割り当てることで、このプログラムを完全に並列化することができる。

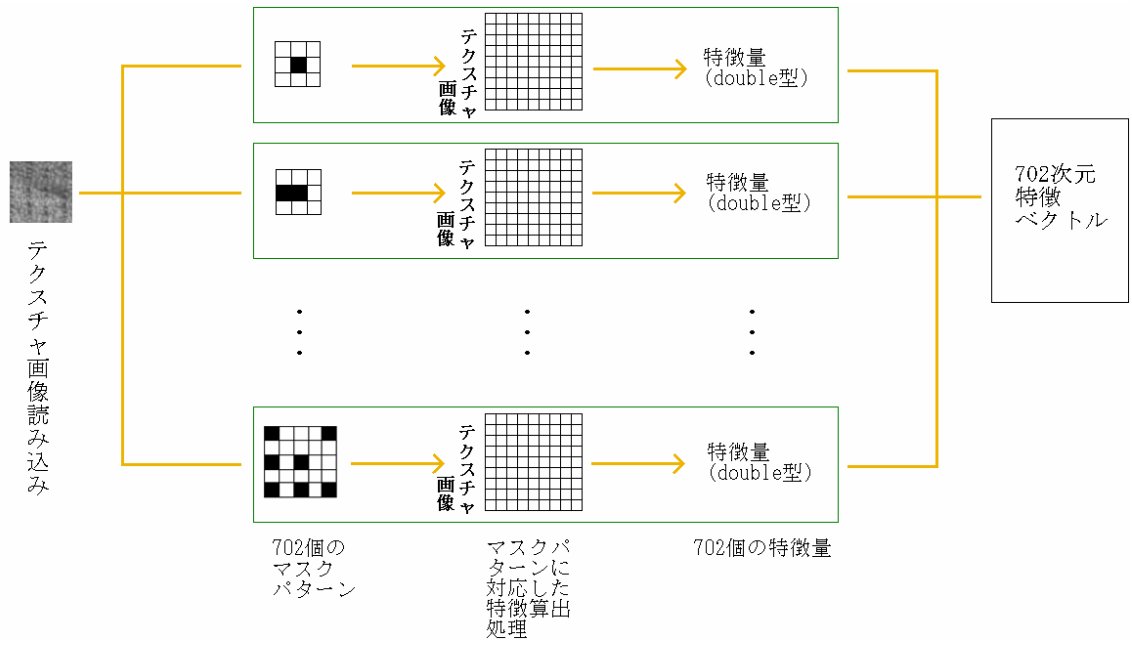


図 14：特徴抽出プログラムの処理の流れ

また、一連の計算の負荷はどの画像でも大きな違いはないが、障害に備えるため動的サイクリックスケジューリングで行なう。ブロック分割でスケジューリングした場合、もし 1 台のノードがビジー状態になり、極度に処理が遅くなってしまったりすれば全体の処理時間がそのノードに引っ張られて並列化する意味がなくなってしまうからである。

5.1.2 テクスチャ識別プログラムの並列化

複数のテストサンプルを識別し、正しい識別結果を得られた確率を識別率とする。1 枚のテストサンプルを読み込み、クラスを識別するまでの一連の処理は全て独立である。そこで、複数あるテストサンプルの処理をノードに割り当てることで並列化する。

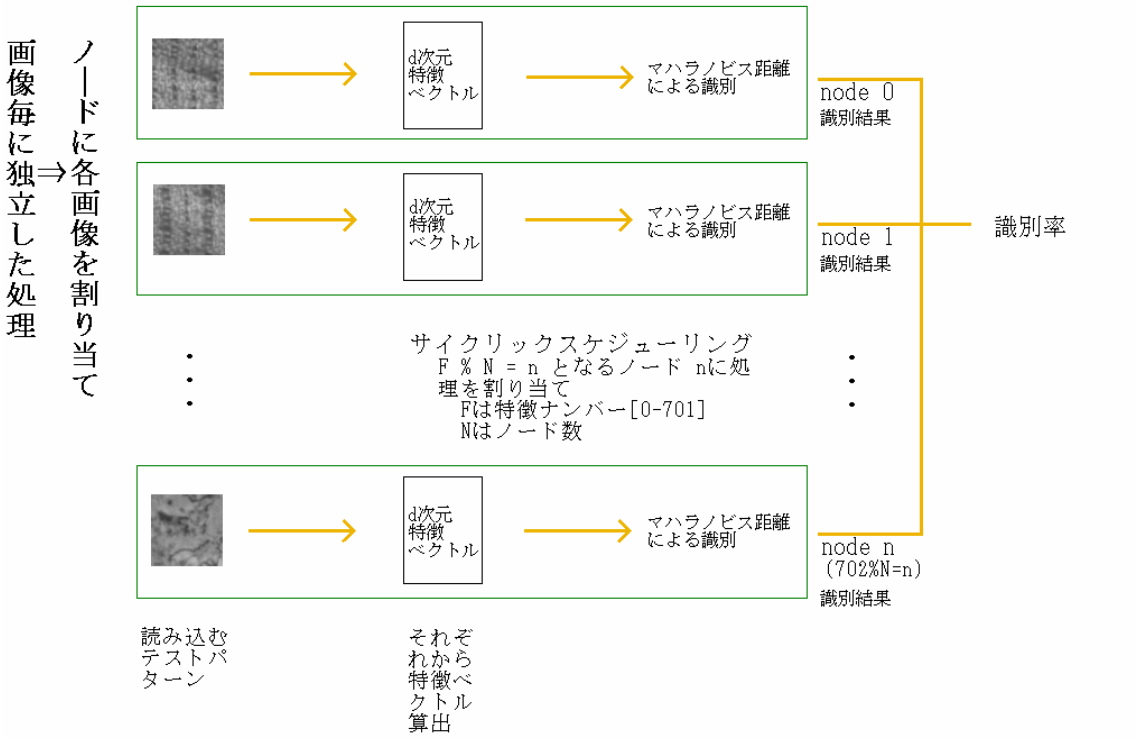


図 15：テクスチャ識別プログラムの並列化

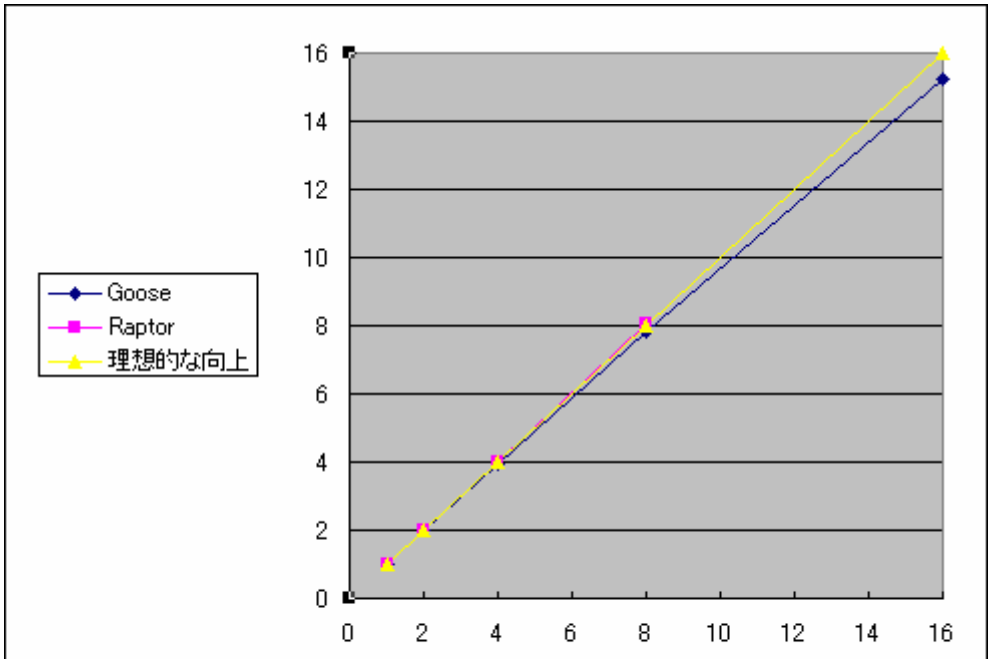
5.2 実験

5.2.1 実験条件

今回実験で用いたのは、Web 上で公開されているテクスチャ識別課題のセット”Outex”[11]の中の Test suite ID:Outex_TC_00001 である。この実験セットには 1 クラスに 88 枚、24 クラスの合計 2112 枚のテクスチャが入っている。それぞれの画像は 64x64 サイズのグレースケール画像である。学習サンプルとして 1 クラス 50 枚、テストサンプルとして残りの 1 クラス 38 枚の画像を読み込んだ。

5.2.2 特徴抽出プログラム

Goose、Raptor でそれぞれ実行した。結果を図 16 に示す。Goose の 16 台実行時にやや速度向上率が下がっているが、ほぼ理想的な並列化効果を得た。



Goose	実行時間（単位:分）	343.36	171.77	86.41	44.01	22.57
	速度向上比	1	1.99	3.97	7.8	15.21
Raptor	実行時間（単位:分）	84.26	41.9	21	10.49	–
	速度向上比	1	2.01	4.01	8.03	–

図 16：実行結果

5.2.3 テクスチャ識別プログラム

識別に用いた 912 枚(1 クラス 38 枚)のテストサンプルの識別精度を図 18 に示す。なお、x 軸の特徴数は、特徴選択プログラムで選択した特徴ベクトルの次元数である。

特徴ベクトルの次元を増やすにしたがって識別精度は上昇し、特徴ベクトルの次元数が 20 になると 87.5%で最高の精度を得た。しかし、それを超えて特徴次元数が多くなると、精度は徐々に下がっていった。なお、特徴次元数を 50 以上にすると、値は不正になり、全てのテストサンプルの識別に失敗した。

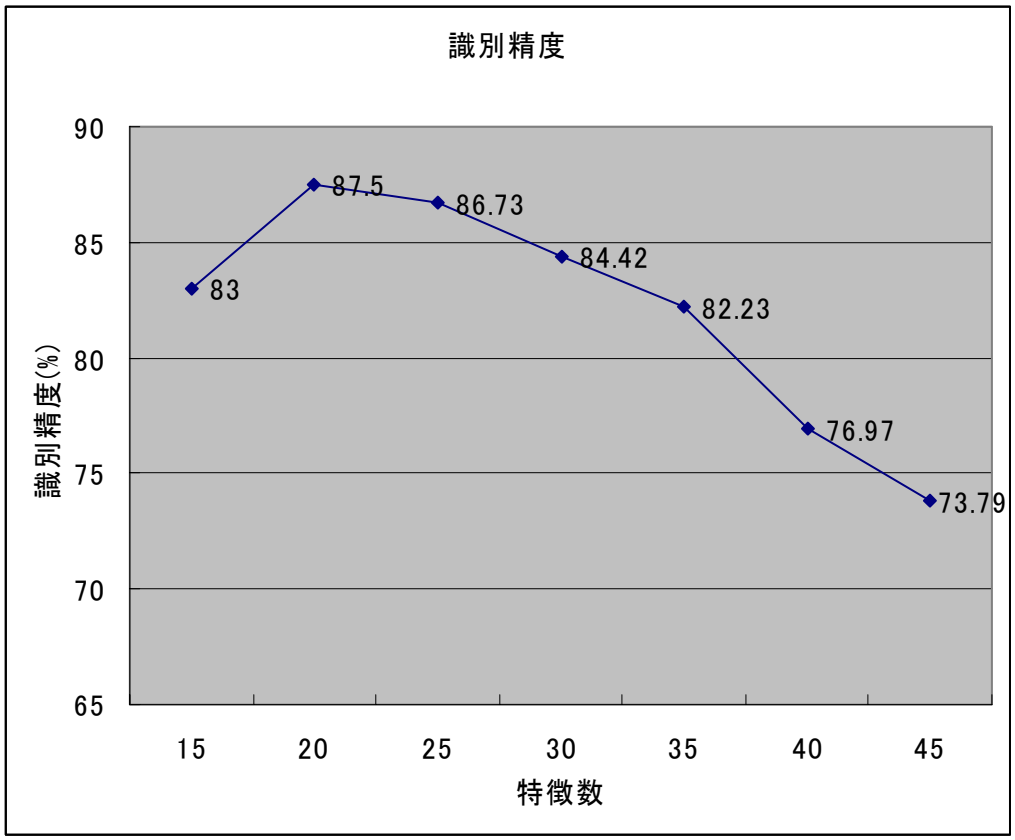
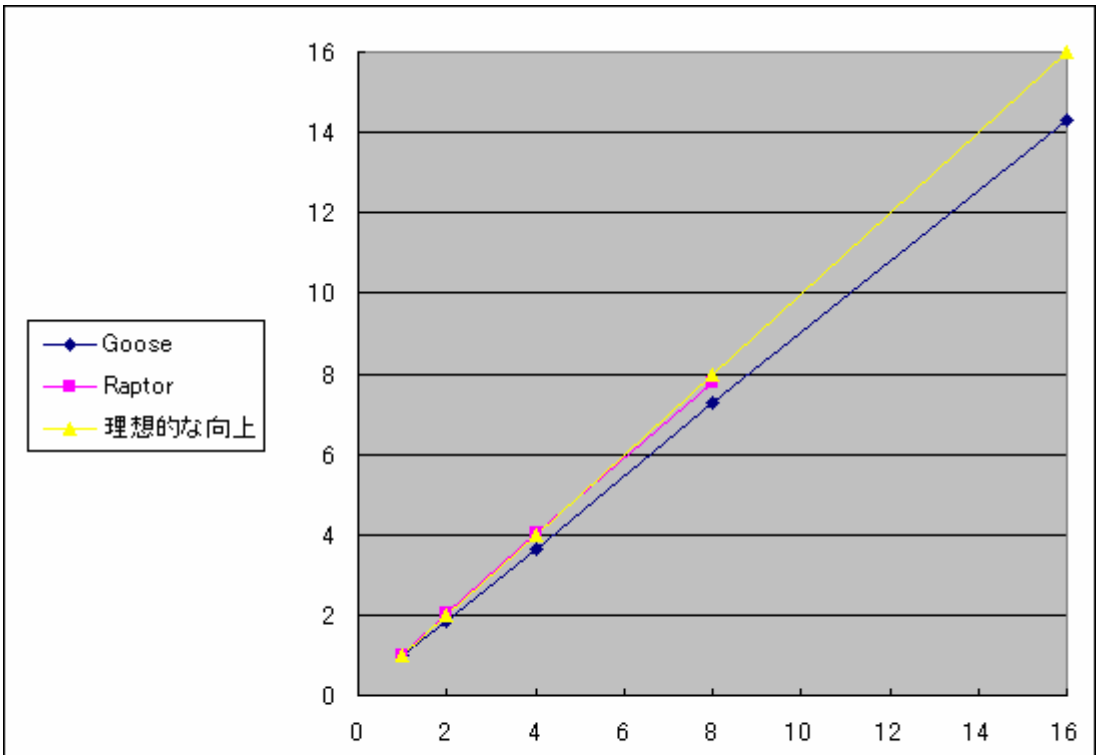


図 17：テストサンプルの識別精度

Goose、Raptor それぞれのクラスタで実行した結果を図 18 に示す。ほぼ実行ノード台数に比例した速度向上が得られている。



Goose	実行時間（単位:秒）	286.38	156.5	78.45	39.22	20.3
	速度向上比	1	1.83	3.65	7.3	14.32
Raptor	実行時間（単位:秒）	89.59	44.35	22.25	11.52	–
	速度向上比	1	2.02	4.02	7.77	–

図 18：実行結果

5.3 考察

5.3.1 特徴抽出プログラム

図 16 に示すとおり、ほぼ理想的な速度向上が得られた。いくつか理由が挙げられるが、1 つは処理のほとんど全ての部分を並列計算していることである。5.1.1 項で述べたように、複数の学習サンプル画像を読み込み処理するがそれぞれの画像に対する一連の処理に依存性が全くないために同期を取ることもなく完全に並列化できた。

2 つ目には計算量に対する通信の割合が少ないことが挙げられる。今回の実験では 1200 枚の 64x64 画像ファイルを読み込み、1200 個の 702 次元特徴ベクトルを書き出している。かなり大きなデータ通信を行なっているが、計算部分がさらに大きいため、相対的に通信部分の占める割合を少なくすることができた。

5.3.2 テクスチャ識別プログラム

特徴数を 50 次元にして識別を行うと、マハラノビス距離の算出に失敗してしまう。これは 4.3.1 項で述べたヒューズの法則とはまた別の問題である。今回用いたテクスチャセット

は、1 クラスあたり 88 枚の画像を持つが、1 クラスあたりの学習サンプルとして読み込んだ画像の数は 50 枚である。

特徴次元数が学習サンプルの数以上になると、識別関数によっては識別そのものが正常動作できなくなってしまう。今回用いたマハラノビス距離を用いた識別関数も、特徴数が学習サンプル以上の数になると完全に識別に失敗してしまうものであった。

マハラノビス距離の算出には、クラス共分散行列を用いる。特徴が 50 個であれば、共分散行列は 50x50 の行列になる。データ数と変数(この場合は特徴の次元)の数が等しいことになる。しかし、共分散行列を計算する際、データ数が変数よりも大きいことは必須条件である。この条件を満たさないと、共分散行列は必ず一時従属な行列になってしまう。これは特異行列とも呼ばれるもので、逆行列演算を行なうことができない行列である。

マハラノビス距離を用いた識別関数では、計算に用いるのはクラス共分散行列そのままではなく、その逆行列である。したがって、特徴数がクラスあたりの学習サンプル以上の数になると逆行列演算が行なえず、識別に失敗してしまう。

また、特徴ベクトルが 20 次元を超えたころから識別率が下がっていくのは、4.3.1 項で述べたヒューズの法則の問題が関係している。例えば 45 次元の特徴ベクトルを識別に用いたとき、データ数>変数であるから、識別関数は正常に動作する。しかし、特徴の信頼度が下がり、識別精度が下がってしまっている。

ほぼ理想的な速度向上が得られた理由は、ほとんどの処理を依存性なく完全に並列化できたこと、並列部の処理量に大きな偏りがなく、負荷均衡がうまく取れていたことなどが挙げられる。

6. おわりに

本研究では、本研究室の SCore 型 PC クラスタ上で OpenMP によってエッジ保存スムージング雑音除去プログラムとテクスチャ解析・識別プログラムを並列化し、実行した。その結果、エッジ保存スムージングでは 3%の確率で雑音を含む 2304x1728 画素のカラー画像の雑音除去で、16 台実行では約 10 倍の並列化効果を得た。また、テクスチャ解析では、特徴抽出、特徴選択、テクスチャ識別の 3 つのプログラムを作成し、内特徴抽出とテクスチャ識別プログラムを並列化して実行した。特徴抽出では 16 台で 15 倍強、テクスチャ識別では 14 倍強の並列化効果を得た。

今後の課題としては、まず、特徴選択プログラムの並列化が挙げられる。このプログラムは依存性がかなりあるプログラムであり、アルゴリズムや変数の扱いをかなり慎重に検討する必要がある。

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いたします。また、本研究に関して貴重な助言や励ましを頂きました的場督永氏、林雅樹氏、池上広済氏、梅原直人氏、及び研究室の皆様に心より感謝いたします。

参考文献

[1]PC Cluster Consortium
http://www.pccluster.org/
[2]OpenMP
http://www.openmp.org
[3]南里豪志,天野浩文：OpenMP 入門(1)、九州大学情報基盤センター研究部、2001.
http://www.cc.kyushu-u.ac.jp/RD/watanabe/RESERCH/MANUSCRIPT/KOHO/OpenMP/openmp0109.pdf
[4]南里豪志,渡部善隆：OpenMP 入門(2)、九州大学情報基盤センター研究部、 2002.
http://www.cc.kyushu-u.ac.jp/RD/watanabe/RESERCH/MANUSCRIPT/KOHO/OpenMP/openmp0201.pdf

[5]南里豪志：OpenMP 入門(3)、九州大学情報基盤センター研究部、2002.
http://www.cc.kyushu-u.ac.jp/RD/watanabe/RESERCH/MANUSCRIPT/KOHO/OpenMP/openmp0209.pdf
[6]誰にでもわかるOpenMP
http://www.ulis.ac.jp/~hasegawa/DELL/OpenMP.html
[7] Stece Oualline 著, 望月康司 監訳, 谷口功 訳, C 実践プログラミング 第 3 版, オライリー・ジャパン, 1998.
[8] 田村秀行, コンピュータ画像処理, オーム社, 2002.
[9] 石井健一郎, 上田修功, 前田英作, 村瀬洋, わかりやすいパターン認識, オーム社, 1998.
[10] 豊田崇弘、長谷川修、テクスチャ識別のためのマスクパターンによる特徴抽出法,2004情報処理学会研究報告,Vol.2004,No.91,pp.77-84, 2004.
[11] T. Ojala, T. M¨anp¨a¨a, M. Pietik¨ainen, J. Viertola, J. Kyll¨onen and S. Huovinen, “Outex-New framework for empirical evaluation of texture analysis algorithms, ” Proc. ICPR, vol.1, pp.701— 706, 2002. (http://www.outex.oulu.fi/outex.php)
[12] 長谷川修, 栗田多喜夫, 坂上勝彦, 大津展之, “高次相関特徴によるテクスチャ解析の試み”, 第55回情報処理学会 全国大会論文集,vol2, pp.258-259, 1997
[12] R. O. Duda, P. E. Hart and D. G. Stork, John Wiley & Sons, “Pattern classification”, 2000.
[13] 平野茂樹, 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, 2004
[14] 前田大輔, PC クラスタ上での OpenMP 並列プログラミング(Ⅱ), 立命館大学理工学部情報学科,卒業論文,2004

[15] 林雅樹, PC クラスタ上での有限要素法によるカルマン渦の並列化, 立命館大学理工学部情報学科, 卒業論文, 2003
[16] 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