内容概要

本研究では、ラベリング処理の並列化による、処理の高速化を図るアルゴリズムを作成し、それを C 言語で記述されたプログラムで実装し、正常に動作するかの検証を行った。

検証を行う為に 10×10 配列を画素集合として定義し、配列の数値を変更することで、2 値化処理を行われた画像を定義したものを、検証パターンとして用意した。その用意した多くのパターンに対して、並列ラベリング処理を行う事で正しく処理が行われているかどうかを検証し、全てのパターンで正しく処理できたことを確認した。

本論文では、ラベリング処理とその処理内容について解説した後、並列ラベリング処理とその処理内容について、また実際にプログラムを動作させた際に、どの様な問題が発生したかという事とその問題に対しての改善を行ったかという部分を解説する。

目次

図目次

1. はじめに

画像処理は 1960 年代より、宇宙探査用の人工衛星画像の画質改善や画像復元への需要が高まり、研究が行われてきた。その後、CT スキャンなどにも用いられるコンピュータの断映像の復元技術の登場で、医学分野と画像処理において多大な影響を及ぼした事により、研究が盛んになった。近年では情報処理技術の発展に伴い、様々な分野で画像処理は用いられるようになっている。この背景から、画像処理に関しての研究は非常に多く、技術が発展するに従い、以前から膨大なデータ量を処理する必要があった画像処理は、処理を行うデータ量が増大している。また、車載カメラの映像による車間距離と車後部の安全確認や、暗視装置や赤外線カメラといったリアルタイムにより近い処理を行う必要がある画像処理技術がもとめられた結果、更なる高速化を行う必要がでてきた。

画像処理の中のラベリングは、画像処理のノイズ除去やエッジ検出といった基本処理の中でも処理時間が長く、コンピュータに対象物を認識させるうえで欠かすことはできない。そのため、高速化を図り、アルゴリズムや処理を行う範囲の改善等による、多くの研究が行われている分野である。しかし、その中でも並列で動作させることで、処理時間を短縮を図る研究はあまり多くない。そこで本研究では、実際に並列で動作するためのラベリングアルゴリズムを作成し、検証を行う。また本研究では、現在本研究室で行っているガラス欠損検出システム開発や甲骨文字認識研究のラベリング処理部に応用することが目的である。その上で、実際にハードウェア上での並列動作を確認するのではなく、まずアルゴリズムの検証を行うという上で書き換えとデバッグがより容易な、ソフトウェア上の C 言語での開発を行う。今後は、実際に並列での動作を行うためにハードウェア上での実装を検討していく。

ラベリングは大きく分けて 3 種類の手法がある。1 つ目は、輪郭追跡である。初めに、ラベル付を行いたいグループと行わないグループの境界線を抽出を行う。次に、境界内のラベルを付けたいグループの外側にラベルを割り振り、グループ内側へと同じラベルを伝播させることでラベリングを行う手法である。この手法は補正の必要はないが、境界外側のラベルだけへのラベル付けは、画素グループが大きい程必要な処理が多くなり、複雑な画素集合に対してかかる処理時間が非常に大きくなる。また、並列化するとラベルが被らない様に、違うラベルでのラベル付けをそれぞれ行わねばならず、その補正のために新たな処理を追加しなければならず、並列化には向いていない。2 つ目は、ラベル伝播である。これはシンプルにラベルを付けたい画素を参照した際、ラベルを付けた画素から 4 近傍もしくは 8 近傍で同じラベルを割り振る。そして、現在ラベル番号を割り振っていたラベルグループに全てラベルを割り振り終わったら、最初にラベルを割り振った部分からスキャンを行う処理である。この手法は単純だが、画素グループごとの伝播する方向の最適化ができていないことと全てラベルを付け終わったかの確認をしなければならないことや、画素グループが多いほどその手間の分時間がかることより、並列化のメリットは大きいが処理自体に時間がかかることが問題である。3 つ目はラスタスキャンによる行ごとへのラベリン

グである。この処理は、初めに行の端から端までのラベル付を行った後、列を 1 つずらして先に割り振ったラベルを参照し、それに基づいたラベルをまた割り振っていくことを繰り返す処理である。この処理は後に正しくラベルが割り振られているか確認しなければならないために補正を行わければならないが、先の 2 つの手法の程の処理時間がかからないことや、複雑な処理があまり多くないことから、本研究ではこの 3 つ目の方法を基にしている。

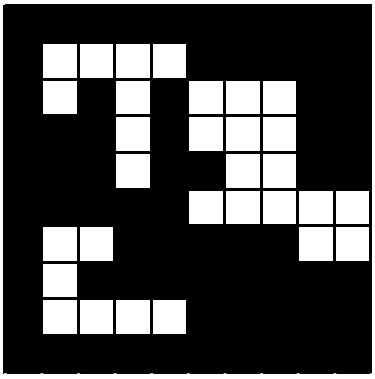
本論文では、まず 2 章で本研究で用いる基本的なラベリングとはどの様な処理かということとその処理内容について解説する。3 章では、本研究内容である 2 方向による並列ラベリングの処理内容と内部動作、特定の画素の並び方が問題となる誤った出力結果を修正するための例外処理の内容について解説する。最後に 4 章で、本並列ラベリングを用いて行った実験とこれから行うハードウェアによる高速化について解説し、実験に対しての考察を記載する。

2. ラベリングとは

2.1 ラベリングの手順

(1)ラベリングの基礎

ラベリングとは、2 値化処理を行われた画像に対して行う、白もしくは黒のどちらかの部分が連続した画素に同じ番号を割り振る処理のことを指す。本論文では白の画素にラベリングを行う。



(a)2 値画像パターン

0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0
0	1	0	1	0	1	1	1	0	0
0	0	0	1	0	1	1	1	0	0
0	0	0	1	0	0	1	1	0	0
0	0	0	0	0	1	1	1	1	1
0	2	2	0	0	0	0	0	1	1
0	2	0	0	0	0	0	0	0	0
0	2	2	2	2	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

(b)ラベリング

図 1.ラベリング

実際に図 1 中の(a)2 値画素集合に白の画素へのラベリングを行うと、白の画素にラベルが割り振られることで、(b)となる。

その用途は通常、同じ番号が割り振られることでグループ化された画素数やその繋がっている画素の幅と高さの特徴量を求めるものであり、その後行う処理の対象となる領域を識別するといったものである。

(2)ラベリングの手順

ラベリングの手順は大きく分けて、２つに分けられる。１つ目は仮ラベル付けである。仮ラベル付けとは、文字通り白の画素に対してラベルを割り振るものである。この処理は各画素に割り振るラベルが最終的に正しいか、誤っているか関係なく割り振っていくものであり、１度だけ行う。しかし、この１度だけの処理では、処理結果が誤っていることが非常に多く、正しいラベルを割り当てる処理を行う必要がある。そのため行われるのが、２つ目のラベル補正である。このラベル番号補正は、仮ラベル付と同じくラスタスキャンを行いつつ、参照地点の 8 近傍のラベルを元に、参照地点のラベルを正しいものへ補正する処理である。

2.2 仮ラベル付け

(1)仮ラベル付け内容

仮ラベル付けは、ラスタスキャン中の参照地点に、ラベルを割り振っていく処理である。

(2)仮ラベル付け参照範囲

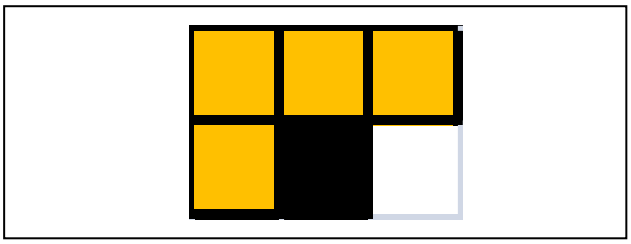


図 2.仮ラベル付け参照範囲

図 2 は仮ラベル付けを行う時の参照範囲である。

参照範囲は橙色で示された 4 箇所である。この 4 箇所の中にラベル 1 以上が割り当てられていた場合は、黒で示された参照地点にその値がラベルとして割り当てられる。参照範囲にラベルが複数存在する場合、その中で最も小さな値が割り振られる。

(3)仮ラベル付け処理

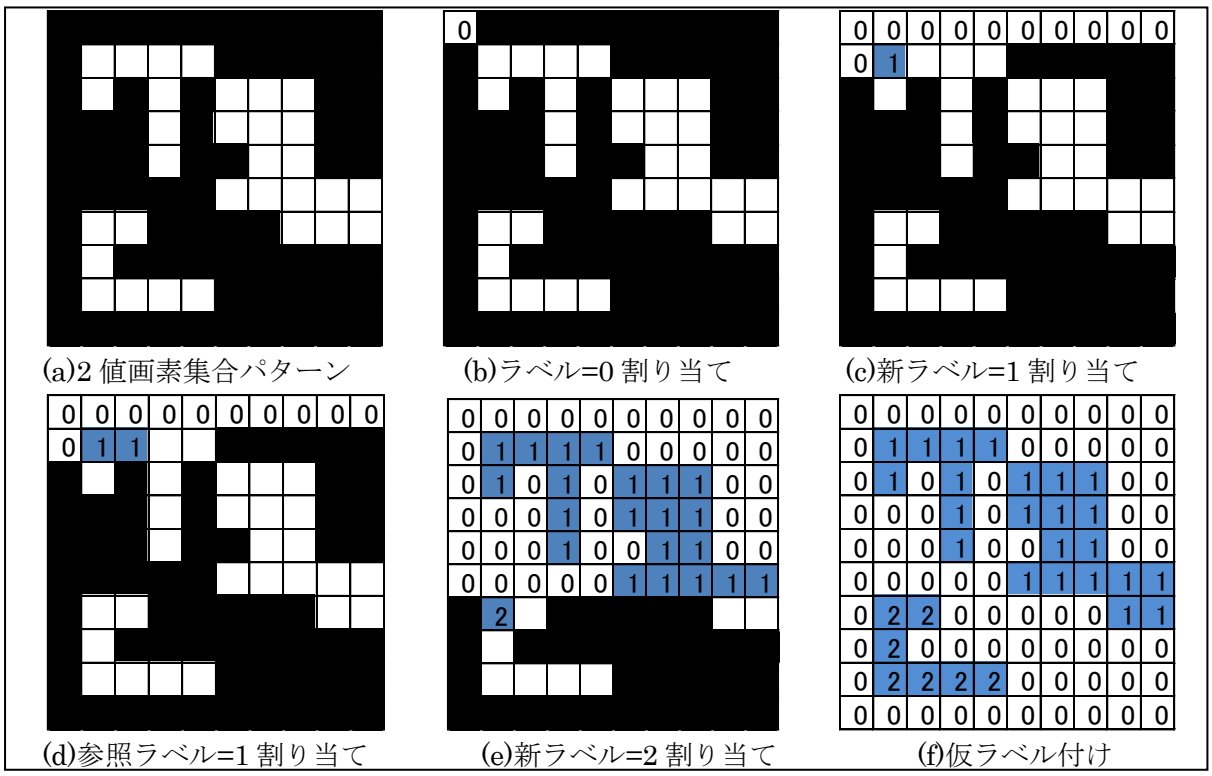


図 3.仮ラベル付け

図 3 は、仮ラベル付け処理である。

ラスタスキャンを(a)の画像を用いて説明する。(a)の左上から x 軸の左端から右端へとスキャンしていく。スキャンの参照地点が右端まで到達した時に x 軸を一つずらし、同じ処理を繰り返す。この処理をスキャンの参照地点が右下に到達するまで行う。

仮ラベル付けはこのラスタスキャンを行い、初めに(b)はラベルを割り当てない黒の画素を参照した時は、0 を割り当てる。以後、黒の画素に対しては必ず 0 を割り当てるが、ラベルを参照する際にラベル 0 はラベルが存在しないものとして扱う。次に、(c)の白もしくは黒のラベルを割り振りたい箇所を初めて参照した時、そこに 1 のラベルをつける。(c)を行った後からは、白の画素に対しては参照範囲にあるラベルに依存する 2 種類のどちらかの処理を行う。1 つ目は、(d)の指定した参照範囲の中にラベルが割り振られている場合である。この時、参照範囲内のラベルで最も小さな値のラベルを参照地点に割り当てる処理を行う。2 つ目は(e)の参照範囲にラベルが存在しない場合である。この場合は、現在まで割り振っていたラベルの値に 1 を足したラベルを割り当てる。(e)では、今まで 1 を割り当てていたので、新しくラベル 2 を割り当てている。

2.3 ラベル補正

(1)ラベル補正内容

ラベル補正は、仮ラベル付けと同じくラスタスキャンを行いながら、参照地点のラベル番号を周囲のラベル番号によって正しいラベル番号に補正する処理である。このラスタスキャンは通常、仮ラベル付けと同じ軌道でスキャンしていく。

(2)ラベル補正参照範囲

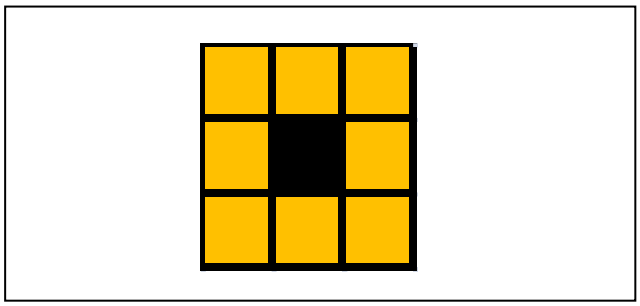


図 4.ラベル補正参照範囲

図 4 はラベル補正の参照範囲である。

参照範囲は橙色で示されており、参照範囲内に現在参照している黒色で示された地点のラベル値より小さな値を持つラベルが存在するとき、参照地点にその値のラベルを割り当

てる。参照範囲内に複数ラベルが存在する場合は最も小さい値を持つラベルを割り当てる。

(3)ラベル補正処理

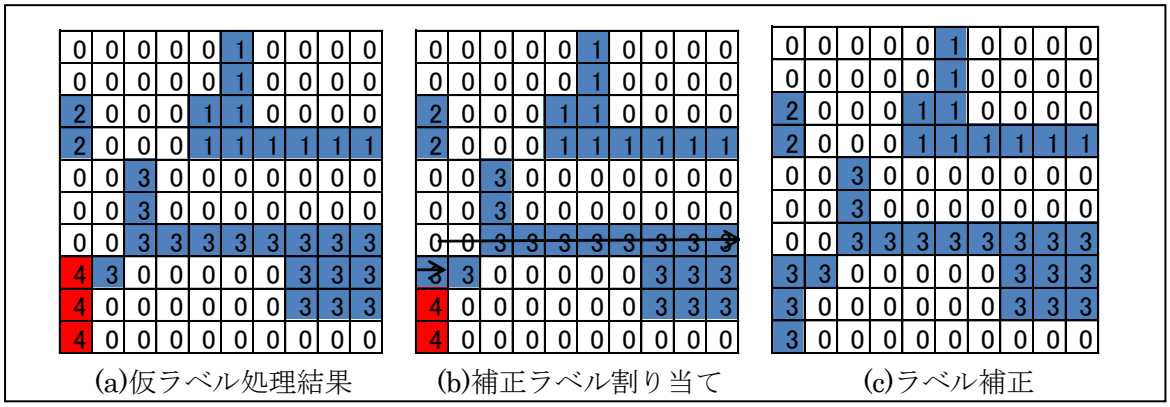


図 5.ラベル補正

図 5 は、8 近傍ラベリング処理中のラベル番号補正処理である。

まず、左上からラスタスキャンを行い仮ラベル付けと同じ軌道で補正を行っていく。(a)のラベルが正しく割り当てられている青で示された部分は、正しいラベルが割り振られている部分である。赤で示された部分はラベルが 4 と割り当てられているが、ラベル 3 と繋がっているため、正しくはラベル 3 が割り当てられなければならない。(b)はラスタスキャン中に 3 箇所ある一番上の誤ったラベル番号を参照した場面である。この時、参照地点の周囲 8 箇所を参照し右にラベル番号 3 と下にラベル 4 が存在することが分かる。この 2 つの中で、最も小さな番号は 3 なので参照地点に 3 のラベル番号を割り振る。これで正しいラベル番号へと補正され、次の参照地点へと移る。この処理を残りを 2 つの誤ったラベル番号に対しても行い、(c)の正しい出力結果を出力することができた。

3. 並列処理によるラベリング

3.1 並列処理によるラベリング内容

本並列処理によるラベリングでは、1 つの画素集合パターンに対して、2 方向からのラベリングを同時に行うことで並列動作を図る。まず、画素集合パターンを上下に 2 等分して、上と下の分割したパターンにそれぞれ番号が被らない、仮のラベル番号を割り振る。次に分割した部分を結合し、結合した部分の番号を統一するため、その部分だけのラベル番号を補正する処理を行う。その後、再度仮のラベル番号を割り振ったのと同じ 2 方向からのラベル番号補正処理を行い、誤ったラベル番号を割り振られている部分を修正し、正しい処理結果を出力するというものである。

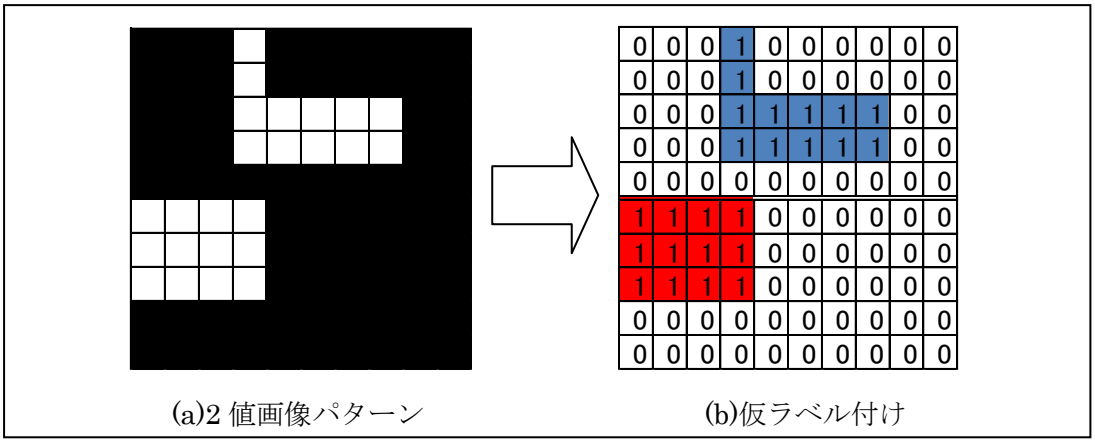
3.2 並列処理による仮ラベル付け

(1)仮ラベル付け参照範囲

本並列処理による仮ラベル付けは、まず画素集合を上半分と下半分に区分することから始める。画素集合を上半分と下半分に区分することで、それぞれに同時に処理を行うことで並列処理を行う。次に仮ラベル付けを行うが、8 近傍処理である故に 8 近傍全てを参照して仮ラベル付けを行っていく訳ではない。

並列処理仮ラベル付けの参照範囲は図 2 と同じで、黒色で示した部分を参照地点とすると、橙色で示した 4 箇所が仮ラベル付けを行うために参照するラベルである。通常の仮ラベル付けと同じ順路でラスタスキャンを行う上で、この 4 箇所は既に仮ラベル付けを行った画素であり、そのラベルのみを参照することで余計なラベルを参照することなく、効率的に仮ラベル付けを行うことができる。また、本研究では C 言語による逐次処理なので、並列で 4 箇所全てを同時に参照し､最も小さいラベルを参照地点に割り当てるのではなく、ラスタスキャンを行ってきた順路通り、左上、上、右上、左の順で参照し、ラベルを参照した場合にそのラベルを参照地点に割り当てる処理を行う。

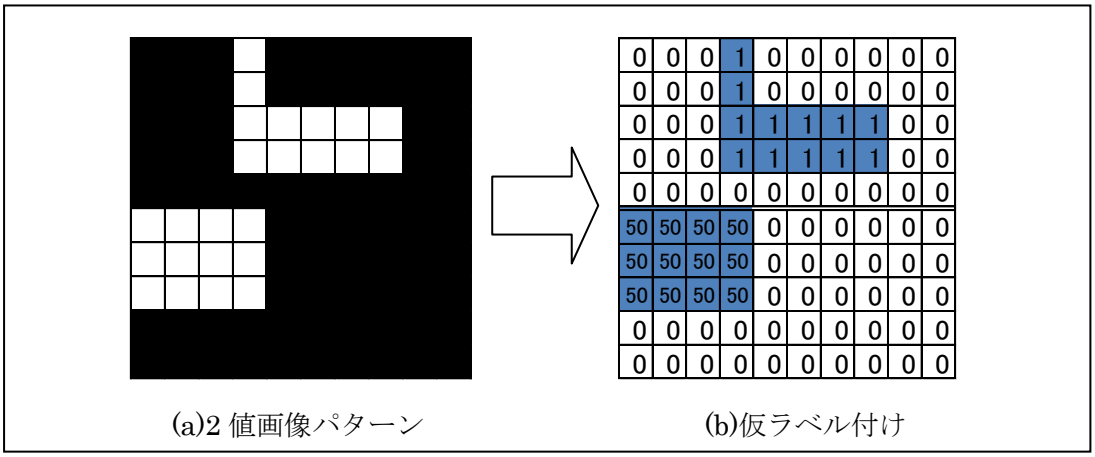
(2)仮ラベル付け番号設定



(a)2 値画像パターン　　(b)仮ラベル付け

図 6.問題仮ラベル付け例 1

仮ラベル付け自体は上半分と下半分、双方に同じラベルを割り当てると図 6 の様に正しい処理を行えない。(b)には 2 つのグループがあるが、同じラベルが割り当てられてしまっているため、実際は違うグループなのを同じグループだと定義してしまっている。正しくは赤で示されたグループに 0 と 1 以外のラベルが割り当てられる必要がある。本論文中では、この下半分の仮ラベルを 50 から割り当て 51、52 と続く様にする。

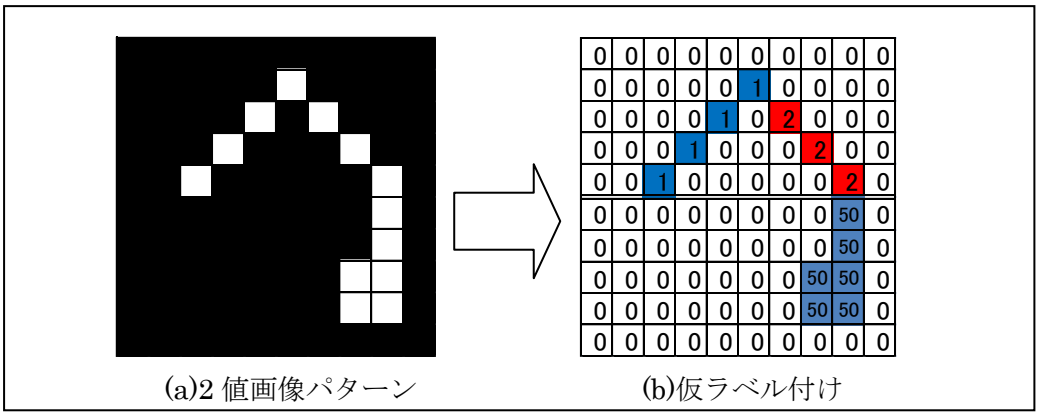


(a)2 値画像パターン　　(b)仮ラベル付け

図 7.正常仮ラベル付け例 1

図 7 を見ると、ラベル 1 の次にはラベル 50 が存在し、その間の 2 や 3 のラベルが存在せず、一見誤った結果が出力されているように見える。しかし、ラベリングは連結している画素のグループが目的であり、ラベルは離れていても違うグループとしての定義には成功しているため、(b)は正しい。

(3)仮ラベル付け順路



(a)2 値画像パターン　　(b)仮ラベル付け

図 8.問題仮ラベル付け 2

後述する並列処理によるラベル補正では上半分と下半分を並列で補正していくが、最終的には結合しなければならない。よって、下半分を上半分と並列で処理するには、上半分のラベルを下半分に伝播させなければならない。この時、下半分のラベルと繋がっている伝播させるラベルが間違っていると、下半分のラベルは全て誤ったラベルを割り当ててしまう。

図 8 は仮ラベル付けの順路と参照範囲を上下逆にしたものであり、このような山形を描く画素集合が上半分に存在した場合、下半分と繋がるラベルが正しく割り当てられない。この問題の原因は、下半分のラベルと繋がっている画素から新しくラベルを割り振っているものである。上から下へと仮ラベル付けを行うと、下半分と繋がっている画素には、周囲を参照した信頼できるラベルを割り振ることが出来るのに対し、下から上へ行うと新しいラベルを割り振っていき、その新しいラベルがそのまま下半分へ伝播される為、結果的に誤ったラベルを割り振っていくことになる。これより、本ラベリングでは必ず上から下への仮ラベル付けを行わなければならない。なお、左と右からではどちら側からといった疑問も残るが、下半分に伝播させるラベルが正しければよく、ラベル補正によってどちらからでも正しいラベルに補正できるので、左からとしている。

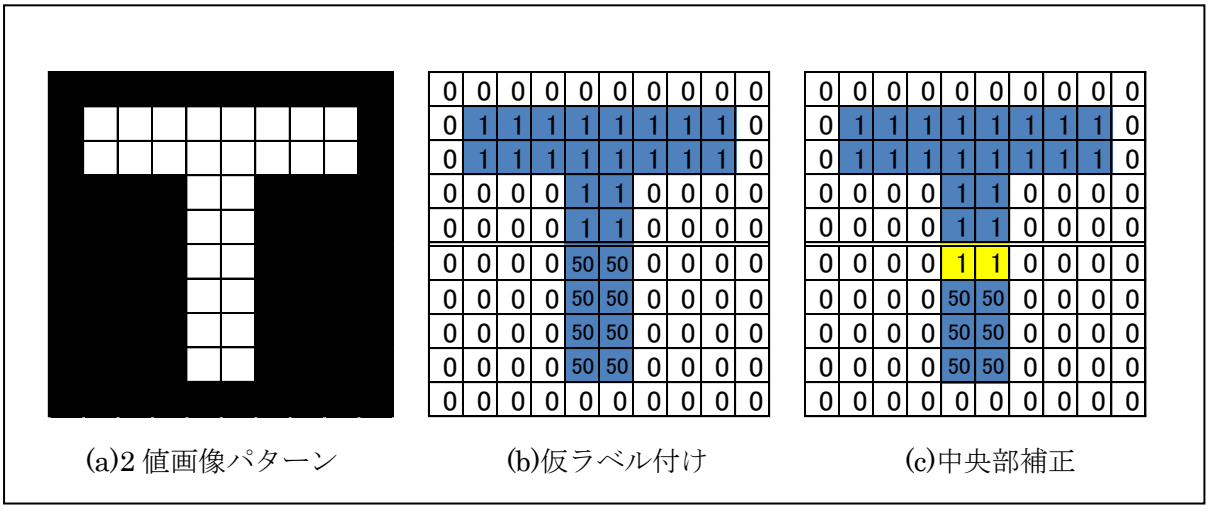
3.3 並列処理によるラベル補正

(1)ラベル補正参照範囲

本並列処理におけるラベル補正では、通常の補正とは異なった手法と順路を用い、またその前に、画素集合の中央部のみへ下半分へ上半分のラベルを伝播させる補正処理を行わなければならない。

補正の参照範囲自体は図 4 の様に通常と同じく 8 近傍を参照し、その中で最も小さな値のラベルを参照地点に割り当てるというものである。

(2)中央部補正



(a)2 値画像パターン　　(b)仮ラベル付け　　(c)中央部補正

図 9.中央部補正

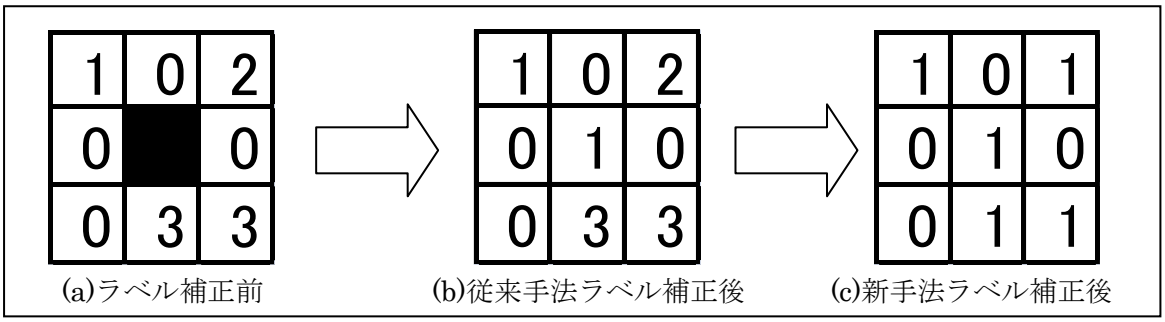
中央部補正は上半分の最下段のラベルを下半分に伝播させる処理であるが、その処理自体は上半分の最下段の x 軸を左から補正していき、次に下半分の最上段の x 軸を右から補正していくという処理である。

図 9 は、中央部補正までの処理である。(c)は補正を行った部分を黄色で示しており、上半分の最下段の画素を用いて、その 1 つの下の下半分の最上段の画素を補正している。こうして伝播させたラベルを用いて、下半分のラベル補正を行う。

(3)新手法によるラベル補正

ラベル補正を行う上で通常のラベル補正手法では、何度もラスタスキャンを行いつつ補正しなければ、正しい処理結果を出力できないパターンが存在する。時間がかかっても構わないのならば、何度もラベル補正を繰り返した後、最後に全体のラスタスキャンを行っ

た際に、補正するラベルが無いことを確認出来れば終了することで、正しい結果を出力することができる。しかし、ラベリングの高速化を行っている本研究では、何度もラベル補正を繰り返していては従来とはほとんど処理時間に変化が見られないので、本並列処理によるラベル補正では新たな手法と同じパターンに対する合計 8 パターンの順路を変更した結果のうち、最もラベルを合計した数値が小さいパターンを正解として出力することで、より早く正確にラベリングを行っている。



(a)ラベル補正前　　(b)従来手法ラベル補正後　　(c)新手法ラベル補正後

図 10.新手法によるラベル補正

図 10 は本並列処理によるラベル補正の流れである。(a)から(b)まではこれまでと同じラベル補正を行うことで、8 近傍中に存在する最も小さなラベルである 1 を参照地点に割り当てているが、新手法では、(b)の後に更にもう一度処理を加える。この処理は、参照地点に割り当てたラベルを参照範囲のラベルに伝播させるというものであり、(c)では(b)の 2 と 3 のラベルが全て 1 に置き換えられている。

(4)従来手法によるラベル補正処理問題例

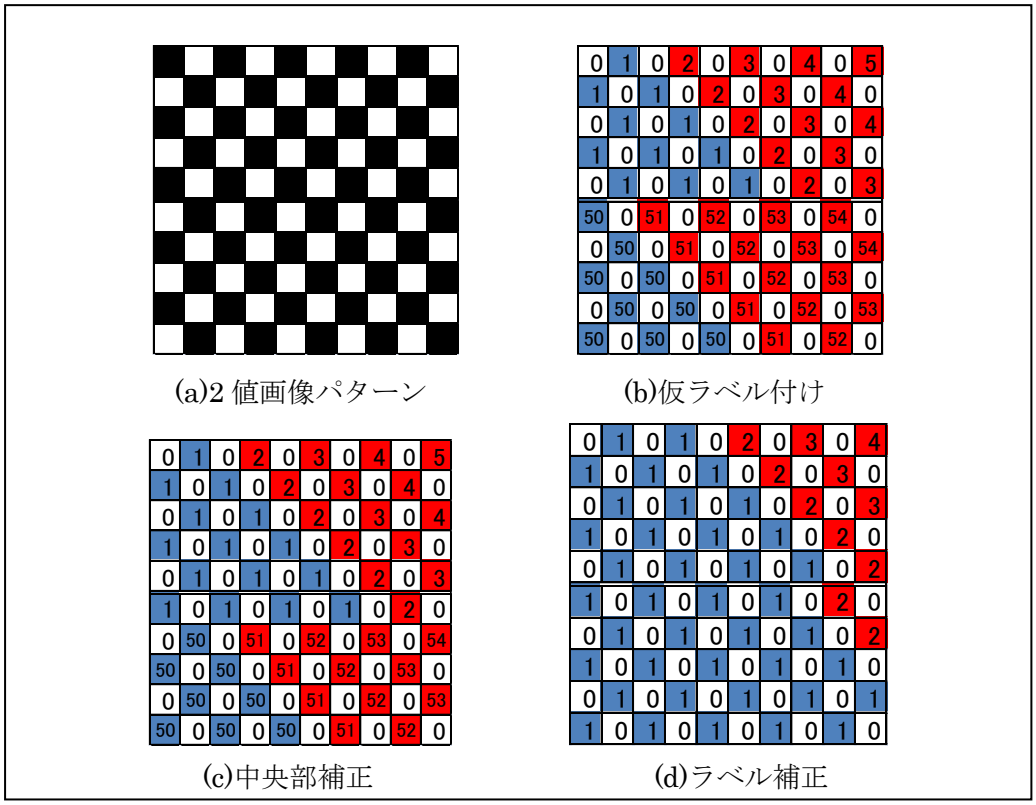


図 11.問題ラベル補正 1

図 11 は、従来の手法でラベル補正を行いラベリングした結果である。(b)を見ると、仮ラベルの段階で誤ったラベルが大量に存在する。存在するだけならばラベル補正で正しい結果を出力できるが、ラベルが大きくなるにつれて、それより小さいラベルに覆われた形になり、1 度のラベル補正では処理できず(b)が(d)になったのに察すると、残り 3 回ものラベル補正が必要なことが推測される。

(5)新手法によるラベル補正成功例

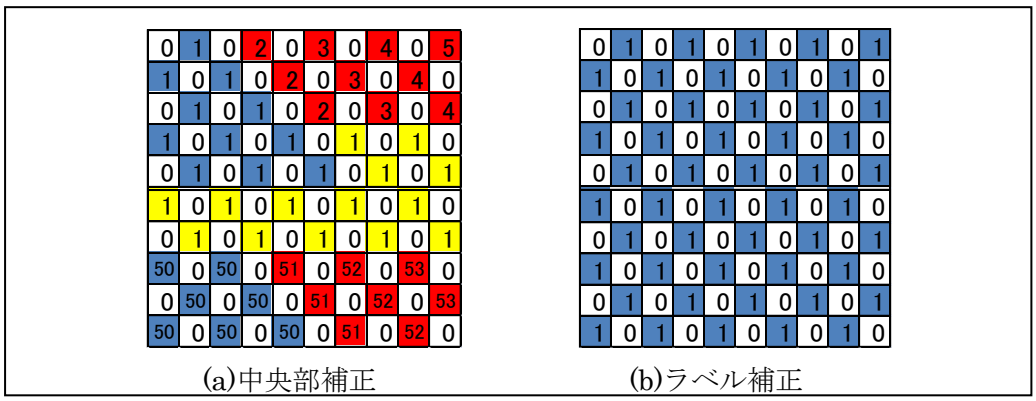


図 12.正常ラベル補正 1

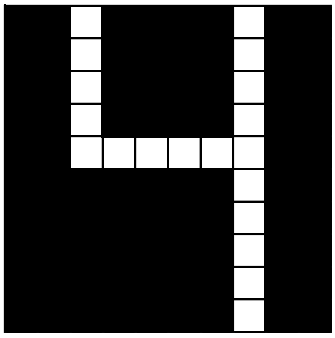
図 12 は新手法で図 11 の(a)に対してラベル補正を行ったものである。

図 11 と異なるのは中央部の補正を行っている(c)からであり、下半分の最上段を補正している時に周りのラベルを巻き込み、参照した画素全てに正しいラベルを割り当てることに成功している。その後、従来と同じ順路で補正を行っていくが、(d)の通り正しい結果を出力することができた。

なぜこの様にうまくラベル補正を行うことが出来るのかというと、端的に言うと今までは参照地点が進む x 軸の 1 列しか補正できなかったのに対し、新手法では 3 列を同時に正しく補正できているからである。また、参照地点に割り振る最小のラベルを利用しているために常に周囲に伝播させるラベルも正しく、伝播させるだけなのでもう一度周囲 8 近傍を参照する必要がなく、短時間でラベル補正を終えることができている。

多くのパターンに対しこの手法を用いることで、より正確なラベル補正を行えてきたがそれでだけでは正しく補正できないパターンもまた多く存在する。次にそれらのパターンに対しては順路の変更によるラベル補正を行わなければならない。

(6)従来の順路によるラベル補正問題例



(a)2 値画像パターン

0	0	1	0	0	0	0	2	0	0
0	0	1	0	0	0	0	2	0	0
0	0	1	0	0	0	0	2	0	0
0	0	1	0	0	0	0	2	0	0
0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	50	0	0
0	0	0	0	0	0	0	50	0	0
0	0	0	0	0	0	0	50	0	0
0	0	0	0	0	0	0	50	0	0
0	0	0	0	0	0	0	50	0	0

(b)仮ラベル付け

0	0	1	0	0	0	0	2	0	0
0	0	1	0	0	0	0	2	0	0
0	0	1	0	0	0	0	2	0	0
0	0	1	0	0	0	0	2	0	0
0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	50	0	0
0	0	0	0	0	0	0	50	0	0
0	0	0	0	0	0	0	50	0	0
0	0	0	0	0	0	0	50	0	0

(c)中央部補正

0	0	1	0	0	0	0	2	0	0
0	0	1	0	0	0	0	2	0	0
0	0	1	0	0	0	0	2	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0	0

(d)ラベル補正

図 13.問題ラベル補正 2

図 13 は、前述の新手法によるラベル補正を用いてラベリングを行った一連の流れだが、正しい処理結果を出力できていない。この理由は単純であり、ラベル補正を行う際に正しいラベルが参照範囲に存在しないので、その誤ったラベルでラベル補正しなければならず、正しい結果を出力することができていないというものである。ここで、参照範囲に正しいラベルが存在しないので、参照範囲を広げようとすると 8 近傍ラベリングではなくなってしまう為にこの手法は使うことができない。なので別の方法で手法でこの問題を解決しなければならない。そこで、順路の変更によるラベル補正を行う。現在は上半分も下半分も仮ラベル付けと同じく左上端からのラスタスキャンにによるラベル補正を行っているが、これを変更して上半分のラベル補正を右下端からのラスタスキャンによるものとする。

(7)順路の変更による改善ラベル補正成功例

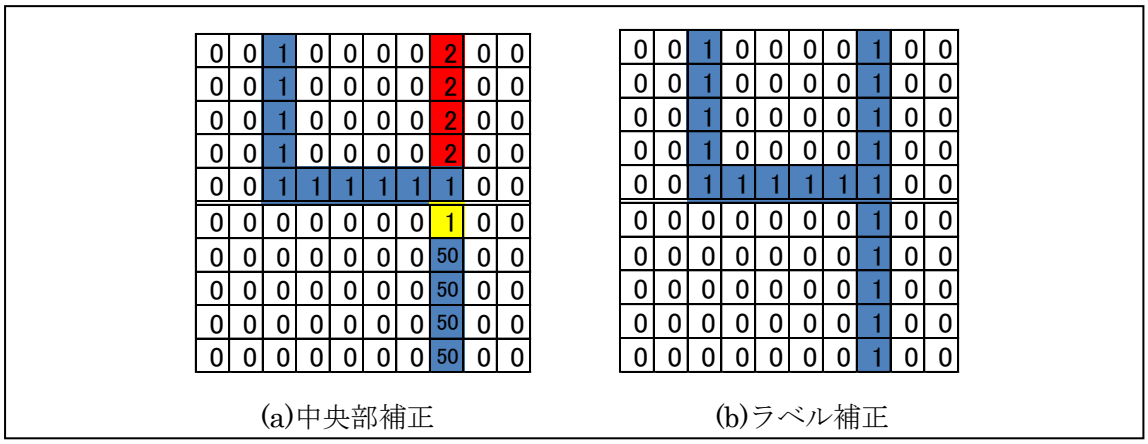


図 14.正常ラベル補正 2

そうすると、(b)の様に正常な結果を出力することができる。この手法により殆どのパターンを正しく補正することができる。だが、実際に順路を変えたラベル補正で行う上で、どのパターンに対してどの順路が最適なのかは実際に全種類を試してみなければ分からない上に、何種類かにしぼって行うと出力結果の信頼性は低い。その為、合計 8 パターンのラスタスキャンによる並列ラベル補正が必要となる。

なぜ合計 8 パターンになるのかというと、上半分のラスタスキャンの開始位置は左上端、左下端、右上端、右下端の 4 箇所あり、また下半分には左上端と右上端の 2 箇所がある。下半分には 2 箇所しかないが、これは下半分の補正に用いるラベルが中央部補正によって得られた下半分の最上段にしか存在しないからである。これにより 4×2 で合計 8 パターンのラベル補正順路を確定することができる。

4. ラベリングの並列化の実験

4.1 成功パターン例

(1)中央部補正に注目したラベリング成功例

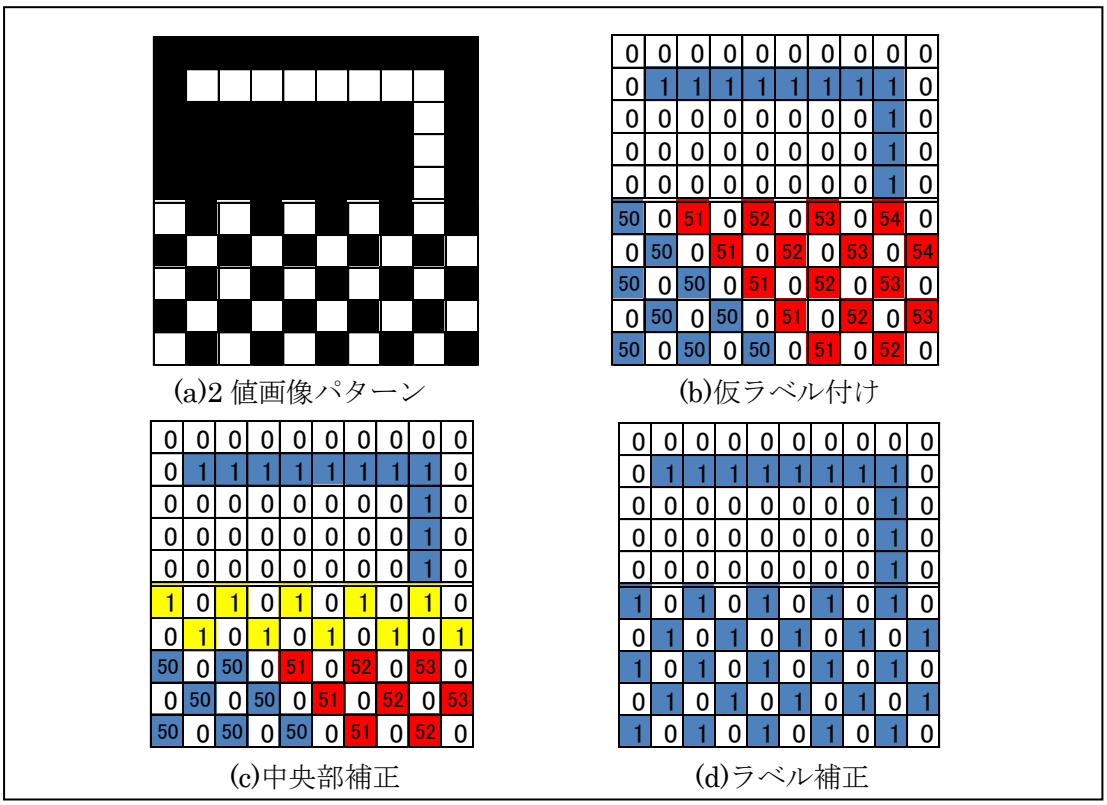


図 15.ラベリング成功例 1

図 15 は、中央部補正に注目したラベリング成功例である。この成功例は中央部補正を単に下半分の最上段の x 軸のみを補正するだけでなく、その前に上の x 軸を補正し下の x 軸の補正は前の補正の逆側から行っていることが重要である。もし中央部補正を単純に左から 1 列しか行わなかった場合、新手法による後述する問題例の様に正しくは補正することができない。ただし、右から中央部補正を行うならこのパターンでならば一応正しい処理結果は出力できる。しかし、上半分の右側から縦に伸びた白の画素が左側から縦に伸びている場合は、やはり正しい結果は出力できない。その為、必ず中央部補正は 2 回行い、行う補正の順路は逆にしなければならない。

(2)補正の順路変更に注目したラベリング成功例

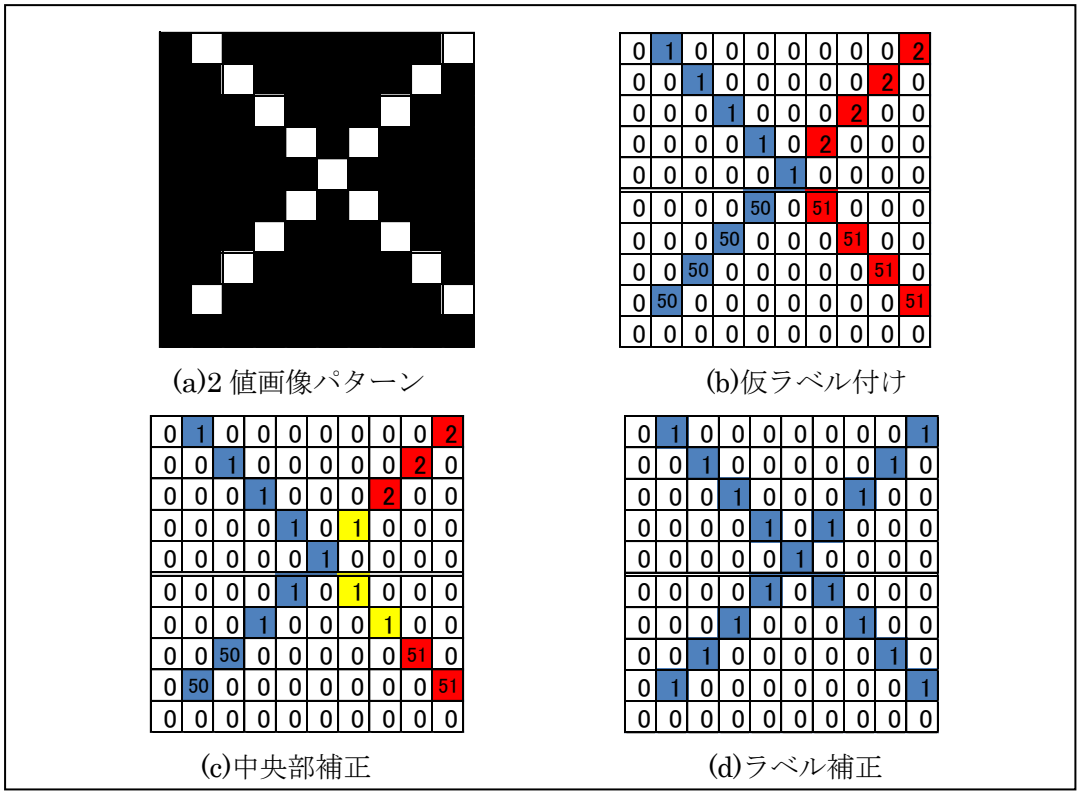


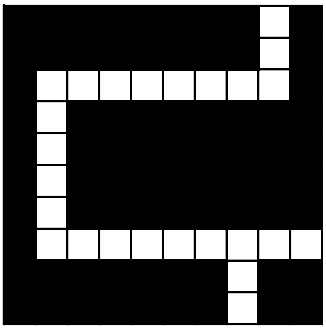
図 16.ラベリング成功例 2

図 16 は、図 15 で解説した縦に伸びているパターンが斜めに伸びているというパターンである。これは図 15 と同じ様に仮ラベル付けと同じラスタスキャンでは正しく補正することは出来ない。ただし、左下端もしくは右下端からラベル補正を行うことで正しく補正できる。この図 16 と図 15 のパターンにより順路の変更により、縦にも斜めにも長く伸びたパターンは従来の順路では正しく補正できないことがわかり、それと同じく順路を変更すれば、これらのパターンを正しく補正できることが分かった。

4.2 例外処理パターン例

これまでに様々なパターンとそれに対する手法を挙げたが、今までに挙げた手法では正しい結果を出力できないパターンも存在する。それらに対しては特定の画素の並び方を検知した場合、例外手法を用いて処理を行うことで正しい結果を出力する必要がある。

(1)上半分に対する仮ラベリングの例外処理



(a)2 値画像パターン

0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1	0
0	2	2	2	2	2	2	1	1	0
0	2	0	0	0	0	0	0	0	0
0	2	0	0	0	0	0	0	0	0
0	50	0	0	0	0	0	0	0	0
0	50	0	0	0	0	0	0	0	0
0	50	50	50	50	50	50	50	50	50
0	0	0	0	0	0	0	50	0	0
0	0	0	0	0	0	0	50	0	0

(b)仮ラベル付け

0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1	0
0	2	2	2	2	2	2	1	1	0
0	2	0	0	0	0	0	0	0	0
0	2	0	0	0	0	0	0	0	0
0	2	0	0	0	0	0	0	0	0
0	2	0	0	0	0	0	0	0	0
0	50	50	50	50	50	50	50	50	50
0	0	0	0	0	0	0	50	0	0
0	0	0	0	0	0	0	50	0	0

(c)中央部補正

0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1	0
0	1	1	1	1	1	1	1	1	0
0	1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	2	0	0	0	0	0	0	0	0
0	2	0	0	0	0	0	0	0	0
0	2	2	2	2	2	2	2	2	2
0	0	0	0	0	0	0	2	0	0
0	0	0	0	0	0	0	2	0	0

(d)ラベル補正

図 17.例外処理必要パターン例 1

図 17 は例外処理が必要なパターンであり、その理由は(b)の時点で下半分と接するラベルが誤っているというものである。中央部補正に用いるラベルが正しくない場合は下半分のラベルは(d)の様に全て誤ったラベルが割り振られてしまう。正しく処理するためには、(b)のラベル 2 を全て 1 にしなければならない。しかし、仮ラベル付けのラスタスキャンが左から行われる以上、この問題は回避できないために例外処理が必要となる。

(2)　(1)へ対しての例外処理内容

図 17 の様なパターンに対しては x 軸一列に同じラベルが並んだ場合、並び始めた場所を記憶し、もしその終端に並んだラベルよりも小さい値のラベルが出現した際は、並んだ数値を終端のラベルの値へと一斉に変化させる処理が必要となる。

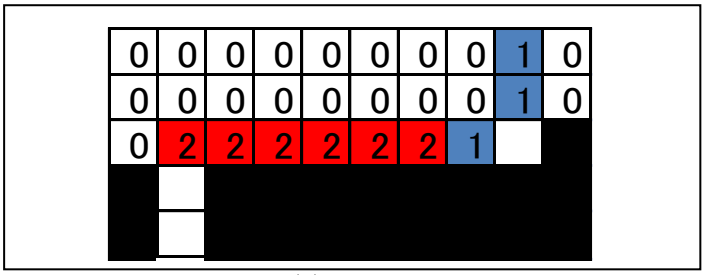


図 18.図 17(b)の上半分パターン

図 18 の場合は仮ラベル付においてラベル 2 が割り振られ始め、その右にもラベル 2 が割り振られた時から、並んだラベルの記憶を始める。ラベル 2 を 6 回割り振った後、次の参照地点で右上の 1 を参照し、参照地点には 1 が割り振られる。ここで、今まで並んだ数値よりも小さな値のラベルが割り振られたことを認識する。

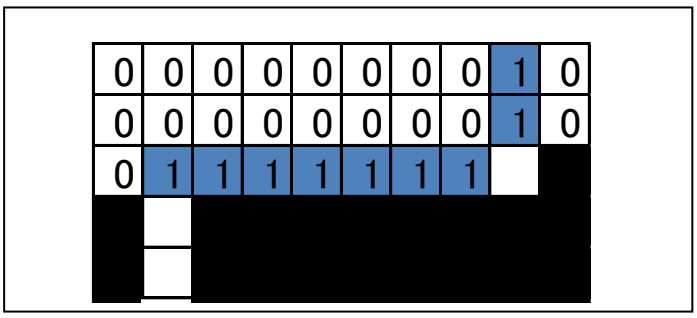


図 19.例外処理結果 1

その時、図 19 の様に左に並んだラベル 2 を一斉に 1 へと変化させる処理を行う。これにより、その x 軸の下の 2 つの白の画素にも図 17 ではラベル 2 が割り振られているが、この処理によって正しくラベル 1 を割り振り結果、下半分にも正しいラベルを伝播でき、正しい結果を出力できる。

(3)下半分で結合するグループへの例外処理

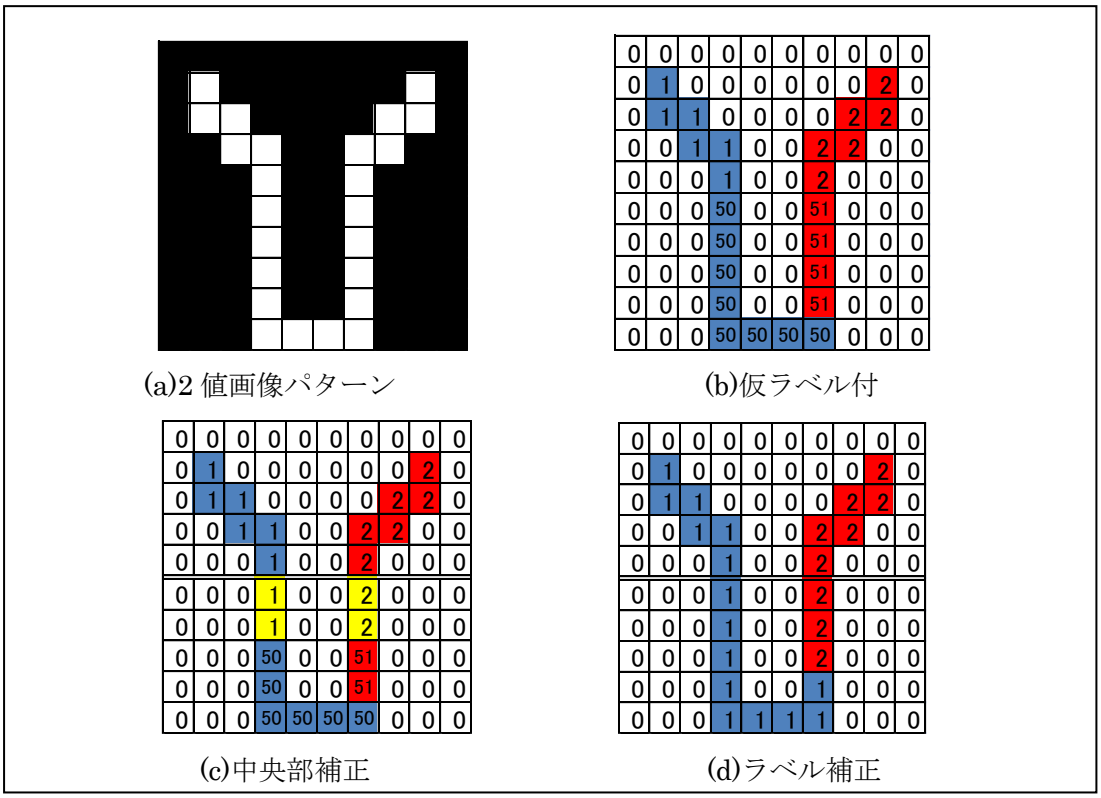


図 20.例外処理必要パターン例 2

図 20 は上半分では違うグループとして定義されているが、下半分で結合しており実際は 1 グループでラベルは全て同じパターンである。このパターンが正しく処理出来ない理由は、(1)と同じだが、決定的に異なる点はラベルを 2 を補正できるタイミングが(a)、(b)、(c)、(d)のどこにもない点である。この様な上半分では離れているのにも関わらず下半分で結合しているパターンへの例外処理は非常に難しく、今まで行ってきた手順のうちのどの手法を変えてもうまく改善しなかった。ただし、このパターンだけならば(a)を 90 度もしくは 180 度回転させた場合の並列処理を行えば、そのうちどれかの出力結果は正しい事が推測できる。だが、将来的により大きな画素集合で本並列処理によるラベリングを行った時、画素中央部に(a)を 90 度ずつ傾けたグループを 4 つ配置されてる場合、前述の並列処理では常にどれか一つは正しくラベリングできないグループが存在するので、結局正しく補正するには何かしらの例外処理を行わなければならない。

(4)下半分で上方向へ伸びるパターン

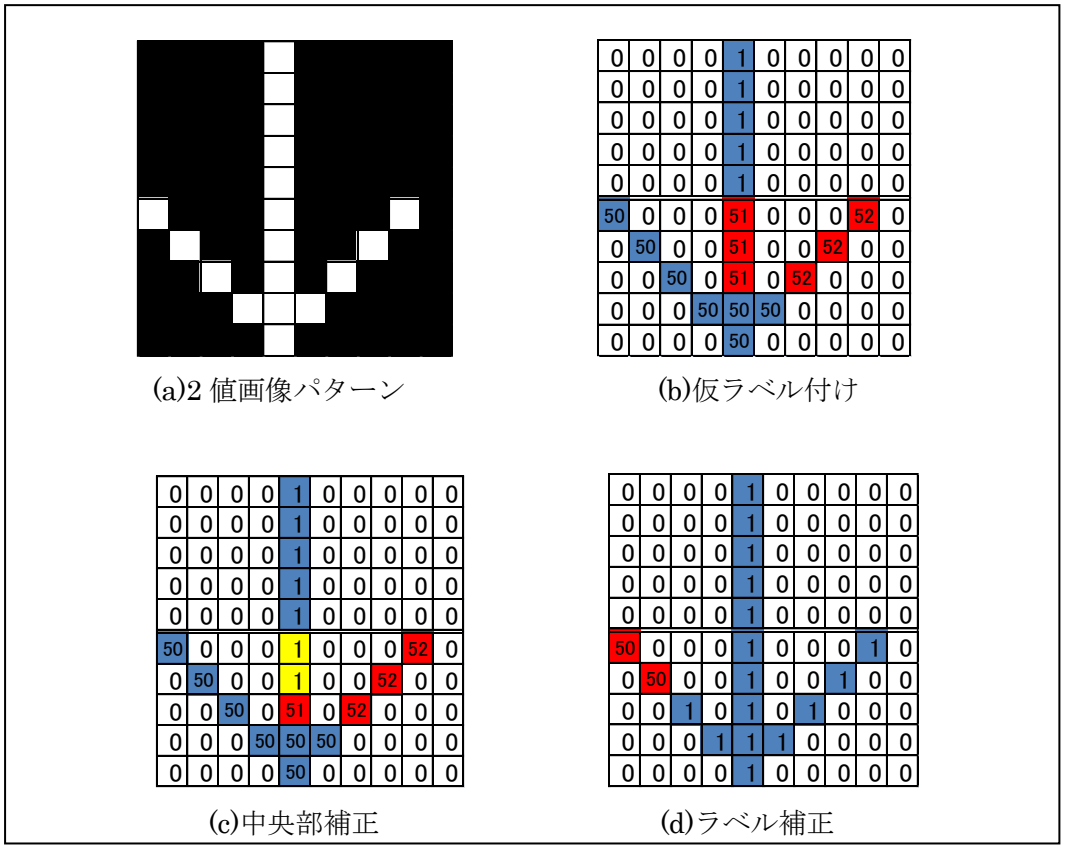


図 21.例外処理必要パターン例 3

図 21 は、下半分でラベルを付ける画素が上半分の補正処理範囲外の距離を取りながら上へと伸びており、下半分のラベル補正は上から下へしか行えないために正しくラベル補正できないパターンである。このパターンも図 22 と同じく、画素集合自体を 90 度もしくは 180 度傾ければ正しく処理できるが、画素集合が大きくなった場合への懸念が残る。

(5)　(3)、(4)に対する例外処理

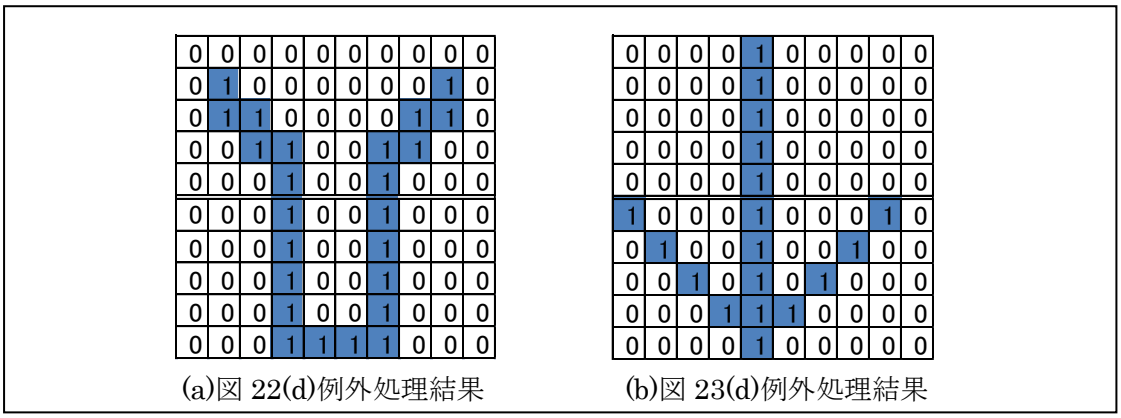
この 2 種類へのパターンに対しての例外処理は同じ例外処理を行う。

ラベル補正を行い上半分と下半分を結合させた後、上半分のラベル補正を左から行った場合は全体画素集合の右下端から、右から行った場合は左下端からのラベル補正を行う。これにより、図 22、図 23 のパターンを正しく処理することができる。

しかし、この例外処理を行うと全体のラスタスキャンを行う事になるため、仮ラベル付けとラベル補正と例外処理の計 3 回のラスタスキャンが必要となる。

仮ラベル付とラベル補正は並列処理で半分ずつ行うので実質 1 回の全体ラスタスキャンとなるが、この例外処理の為に処理時間が約 2 倍になってしまう。また、図 22 や図 23 のパターンが画素集合中に含まれているかは、本並列処理によるラベリング中では検知出来ない。よって、この例外処理は必ず行わなければならない。

(6)図 22 と図 23 への例外処理結果



(a)図 22(d)例外処理結果　　(b)図 23(d)例外処理結果

図 22.例外処理結果 2

図 22 は、図 20(d)と図 21(d)に対する例外処理結果である。

例外処理を行ったことで、赤で示されていた誤ったラベルが正しく補正され、全ての画素が繋がっているためにラベル 1 が割り当てられていることが確認できる。

4.3 ハードウェアによる高速化

現在は C 言語を用いた逐次処理でのアルゴリズムの検討を行っているが、将来的にはハードウェア上で実際に並列処理を行う。現在は全て逐次処理であるソフトウェア上で動作させているが、ハードウェア上で動作させる。それにより、仮ラベル付とラベル補正、ラベリングそのものを並列で動作させることができる。また、ハードウェア量を増やすことにより、現在は 2 方向での処理を行っているが、より並列化された 4 方向や 8 方向といった処理も期待できる。

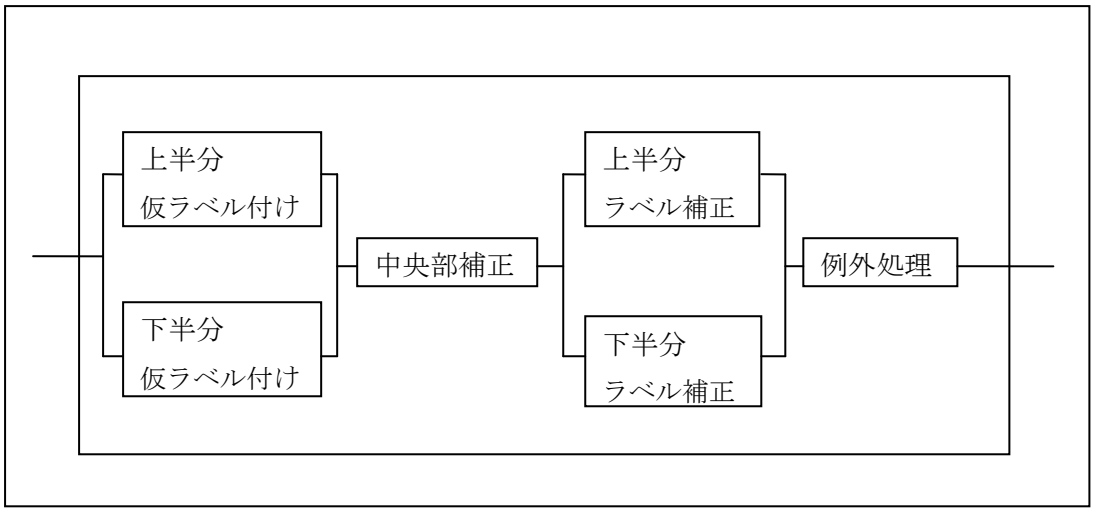


図 23.並列ラベリング内部

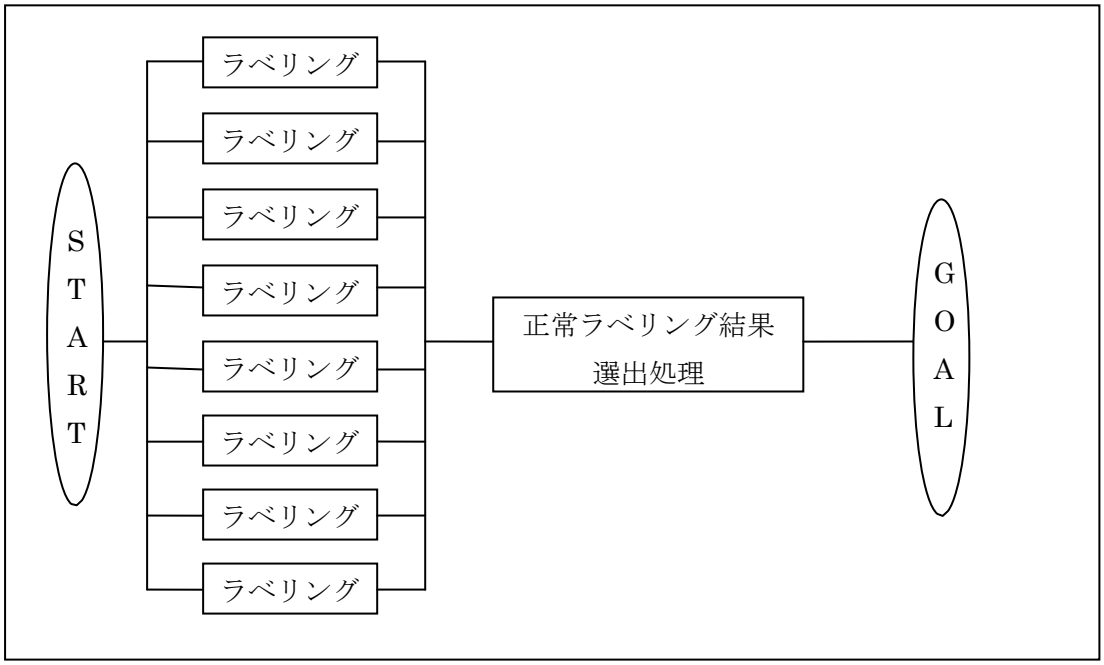


図 24.並列ラベリング

実際に並列処理でラベリングを行うと、内部処理は図 23、図 24 の様になる。

図 23 は、ラベリングの内部ではどの様な処理を行っているかを表している。そして、図 24 はラベリング自体を並列処理で行い、その中から正しい処理結果を選出処理を行う流れを表している。

4.4 考察

本研究である並列処理によるラベリング処理のアルゴリズムの検討の結果、通常より大幅な処理時間の削減が見込める様になった。また、補正不足や誤ったラベルを割り当てた処理結果を出力しない様に、新手法や例外処理の導入したことで、信頼性も大きく向上した。

しかし、まだ改善が見込める点は多い。

1 つ目は、ラベル補正の参照範囲である。現在は 8 近傍全てを見ているが、余計な箇所を参照している可能性があり、将来的にはもっと参照範囲を削減する事での高速化出来る可能性がある。

2 つ目は、全体ラベル補正を行う例外処理である。この例外処理を行う必要が有るため、約 2 倍の処理時間を必要としている。図 22 と図 23 の画素の並び方に対してよりよいラベル補正アルゴリズムが存在すれば、より処理時間を短縮できる。

3 つ目は、8 種類あるラベル補正の中で最もラベル値の合計が小さいパターンを選出する処理時間の短縮である。逐次処理では実際に並列での動作は不可能なので、ハードウェア上で実装した際にアルゴリズムの検討を行わねばならず、その分の処理時間はまだ未知数である。現在は最後の例外処理を行いながらラベル値を記憶、合計したものを比較して最も小さな値を持った結果を最終的な出力結果としているが、これより効率のよいアルゴリズムが存在している可能性があり、それを用いることでの高速化が見込める。

5. おわりに

本論文では、並列処理によるラベリングを逐次処理の C 言語を用いて検討し、新たに取り入れた新手法や例外処理の追加により、より早く正確にラベリングを行うことができるアルゴリズムを作成した。

本研究を通して、従来のラベリングでは正しい処理結果を出力できない部分も存在することや、それに対しての例外処理が必要なことが分かった。また、ハードウェア量を増やすことで別々の処理を並列で実行することができ、それにより処理時間の短縮が図れることも理解した。

今後の課題としては、考察に挙げた 3 つの改善点を見込める手法についての再検討である。また、ハードウェア上で実行させた際の処理結果の選出法や仮ラベル付けとラベル補正範囲の更なる効率化が挙げられる。

謝辞

本研究の機会を与えてくださり、ご指導いただきました山崎勝弘教授に深く感謝します。また、本研究に関して貴重な助言、ご意見をいただきました孟助教授、そのほか高性能計算研究室の皆様に心より感謝いたします。

参考文献

[1] 井上誠喜：C 言語で学ぶ実践画像処理,オーム社,2008.
[2] 内村螢一,上瀧剛共：実践画像処理入門,培風館,2007.
[3] 松山圭輔：FPGA を用いた液晶用ガラス欠損検出システムの高速化,立命館大学理工学研究科電子システム学専攻修士論文,2012.
[4] 岡崎大輔：GPA を用いた液晶用ガラスの欠損検出画像処理の高速化(Ⅰ),立命館大学理工学部電子情報デザイン学科卒業論文,2012.
[5]松崎裕樹：マハラノビス距離を用いた画像判別とラベリングの高速化の実現,立命館大学理工学部情報学科卒業論文,2006.
[6]手塚一雄：はじめての C 言語,技術評論社,2000.
[7]田村秀行：コンピュータ画像処理,オーム社,2004.
[8]大崎紘一,宗澤良臣,神代充,梶原康博：画像認識システム学.共立出版.2005.
[9]松山圭輔,孟林,野尻直人,泉知論,山崎勝弘：FPGA を用いたガラス欠損検出システムの高速化, FIT2013,C-012,2013.