卒業論文

GPU を用いた液晶用ガラスの欠損検出
画像処理の高速化[Ⅱ]

氏名：野村寛

学籍番号：2260070075－1

指導教員：山崎　勝弘　教授

提出日：平成 23 年 2 月 18 日

立命館大学　理工学部　電子情報デザイン学科

内容梗概

本論文の目的は、ガラス基板に存在する欠損を発見する画像処理を、高速化することである。まず、C 言語を用いてラプラシアンフィルタ、ラベリングを行い、ガラスの欠損を発見する。また、TDI と呼ばれるノイズ除去の前処理を行った場合と、行わない場合の比較も行う。その後に、CUDA を用いて GPU に実装し、高速化可能かを検証する。CPU では、TDI を行わない場合は欠損部分を確認出来なかったが、行った場合は欠損の検出を確認できた。CPU 場合、最大で、約 28 倍（TDI128 回）速度向上を得ることが出来た。GPU では、ラプラシアンフィルタのみの実装となった。出力画像において、輪郭を抽出は行えた。少し処理を変えたので、CPU との処理時間の比較を単純に行えないが、約 30 倍の速度向上を得られた。

目次

図目次

表目次

１　はじめに

デジタル画像処理は、1960 年代より、文字認識、物体認識、医療用画像処理、人工衛星からの画像の協調などの分野で進んできた。1970 年代になると、工業用画像処理、衛星画像処理、医療用画像処理の分野が盛んになり、CT（Computed　Tomography：コンピューター断層撮影）装置が生まれた。これにより、脳の断面図を簡単に見られるようになり、医療の画期的な進歩を遂げた。また、リモートセンシング（遠隔探査）の技術により、人工衛星からの地球表面の資源データなどを処理することが可能となった。この技術は、気象情報の画像化や環境汚染の調査にまで利用されている。

そして最近では、工場でも画像処理技術が利用されており、欠陥品の自動検査、産業用ロボットの目などに使われている。この場合、生産ラインで利用されるので、より高速な処理を求められる。そして近年、PC、液晶テレビ、スマートフォンなどの需要が高まってきており、液晶用ガラスの需要も高まってきている。それにより欠損検出も重要となる。
本研究では、液晶用ガラスの欠損検出画像処理の高速化を目的としている。まず TDI により原画像のノイズを軽減させる。次にラプラシアンフィルタをかけて画像内のエッジを検出する。そして、ラベリングを行って画像内の各オブジェクトを識別することにより、欠損を検査する。実験では、まず C 言語を用いて、ガラスの欠損検出実験行う。その次に、GPU を用いて高速化を行う。対象となる画像データは、ケー・デー・イーより頂いた欠損画像である。画像のサイズは 1280×960 ピクセル 8 ビットのｂｍｐファイルで、輝度 0~255 のグレースケール画像である。評価は、TDI 処理の有無での出力画像と速度の比較、CPU と GPU での出力画像と速度の比較とする。
GPU とは（Graphics　Processing　Unit）の略で、画像処理を行うために特化した、専用のプロセッサで、最近のパソコンのソフトは画像処理を多用するものが多く、それらをコマ落ちしないように画面表示するような処理にも使われている。
GPU には、単純な演算器がたくさんあり、並列処理を行う。その１番小さな単位が、スカラプロセッサ（SP:　Scalar　Processor）である。その１つ上にストリーミングマルチプロセッサ（SM:　Streaming Multi-processor)があり、SM には８個の SP がついている。実際に処理を行う１つ処理のことをスレッドといい、その上のスレッド集まりをブロックといい、さらにブロックの集まりをグリッドという。また、GPU には CUDA を用いてプログラミングを行う。CUDA は NVIDIA が提供する GPU 向けの統合開発環境であり、C 言語の文法を一部拡張したものである。２章では、画像欠損発見のアルゴリズムを述べ、３章で C 言語での欠損発見画像処理の検証を示し、４章で GPU と CUDA の概要を述べた後に、GPU 上での欠損発見画像処理の高速化の検証を行う。

２　ガラスの欠損発見画像処理アルゴリズム

２．１　TDI（Time Delay Integration）

TDI は、時間遅延積分方式のことであり、ある画像の画素をずらして複数回撮影し、共通部分の各画素を平均化することによって S／N 比を高める手法である。これにより撮影時に入ったノイズの部分が平均化され、ノイズの影響が少なくなり、他の処理の精度を上げることが可能になる。

図１から図２に例を示す。赤色の部分がノイズである。図１が TDI 処理前、図２（a）が TDI 処理後、図 2(b)が TDI 処理の理想画像である。図 2(a)の緑色の箇所が平均化された部分である。図２（a）を見ると、平均化されノイズが理想に近づいている。

TDI４回の場合、

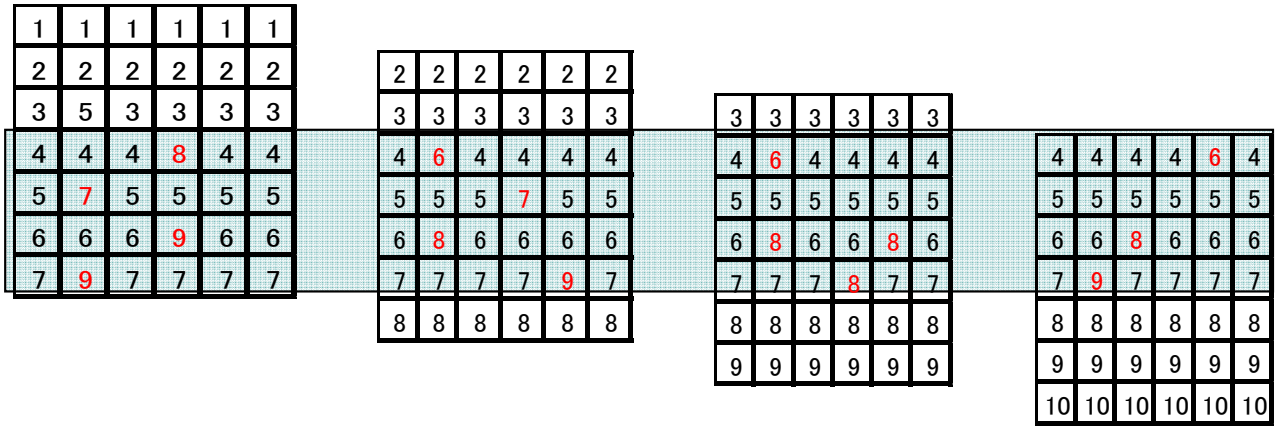


図　１：TDI 処理前

4	5	4	5	4.5	4
5	5.5	5	5.5	5	5
6	7	6.5	6	6.8	6.5
7	8	7	7.3	7.5	7

(a)TDI 処理後

4	4	4	4	4	4
5	5	5	5	5	5
6	6	6	6	6	6
7	7	7	7	7	7

(b)TDI 処理　理想画像

図　２：TDI 処理後と理想画像

２．２　ラプラシアンフィルタ

ラプラシアンフィルタは、注目画素から近傍３×３あるいはそれ以上の空間に、空間２次微分を計算して、画像中の物体の輪郭部分（エッジ）を抽出するフィルタである。注目画素の周りの４近傍、または８近傍にフィルタの係数をかけて、その和を足した結果を注目画素に入れる。それを順番に最後まで行っていく。
演算を次に示し、４近傍の場合のフィルタを図３（a)、８近傍の場合のフィルタを図３（ｂ）処理の流れを図 3(c)に示す。

$$\nabla^2 f(x,y) = f_{xx}(x,y) + f_{yy}(x,y)$$

これを差分形式で表すと、

$$\nabla^2 f(x,y) = f(x-1,y) + 2f(x,y) + f(x+1,y) + f(x,y-1) - 2f(x,y) + f(x,y-1)$$

$$= f(x,y-1) + f(x+1,y) + f(x,y-1) + f(x,y+1) - 4f(x,y)$$

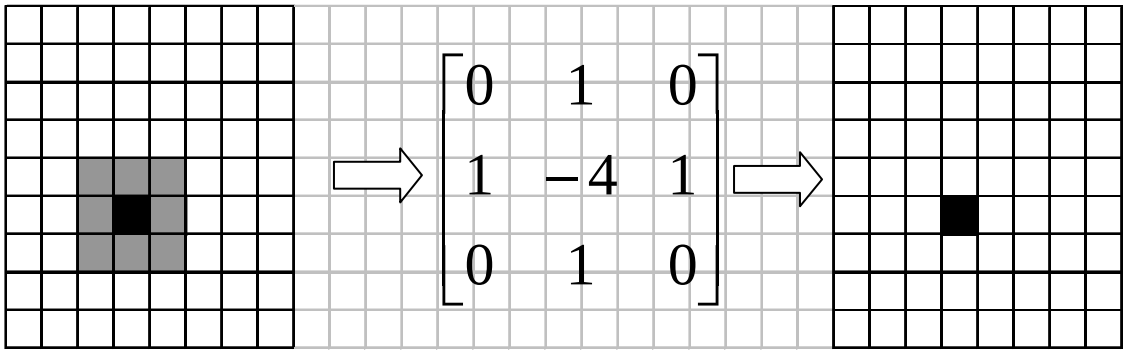
これらを係数で表すと、

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

（a） ４近傍の場合

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

(b) ８近傍の場合



(c)処理の流れ

図 3：ラプラシアンフィルタ

２．３　ラベリング

ラベリングとは、２値化された画像の画素中の同じ連結成分に１つの番号（ラベル）を割りふって、各連結成分に別の番号を割りふる処理である。これにより、各連結成分の特徴量を抽出することでき、肉眼で確認しやすくなる。図４にラベリングのイメージ図、図５から図９に処理の流れの例を示す。

図 ４ の場合、３つの連結成分が存在する。この３つの連結成分にそれぞれ違う番号を付けていく。図 4(a)の水色の成分の黄色い線に囲まれている順番で成分を見ていく。

今回はルックアップテーブル法を用いた。まず左上の画素からスキャンしていき、白の画素を探す。上、左、左上、右上の順で見て、白の画素を発見したときに、その中にラベルが無かったら新しいラベルをつける。図６が初めてラベルが付く時である。次に上、左、左上、右上を見た時に、その中にラベルが存在する場合、そのラベルをつける。これを最後の画素まで行う。図７が、１回目の操作が完了した時である。３と２を１に変更する必要があるので、２週目の操作行う。２週目にラベルがつけられている画素を見ていき、右，下、右下、左下の順に見る。この中に違うラベルがあった場合、それをルックアップテーブルに登録する。図８（ｂ）を見ると、ルックアップテーブルの３が、１に登録されている。この処理を繰り返し、登録する必要が無くなるまで行う。最後に、ルックアップテーブルを参照して、ラベルを貼り代える。図９でラベルが貼りかえられている。

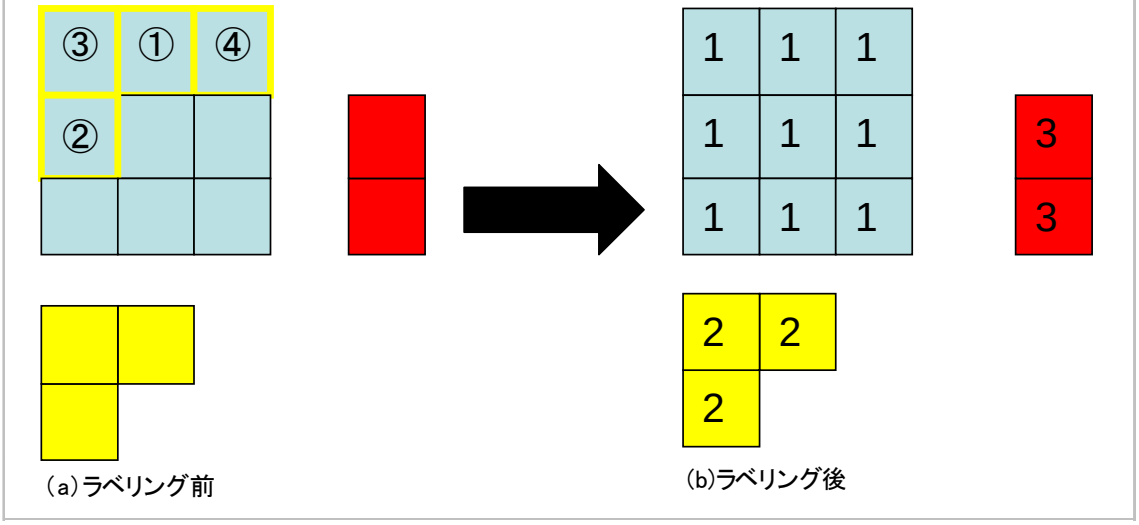
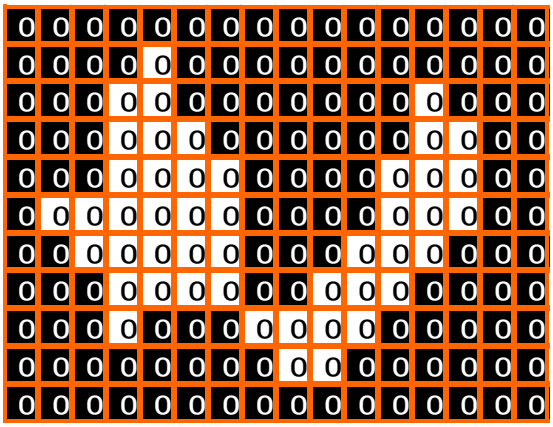


図 ４：ラベリング

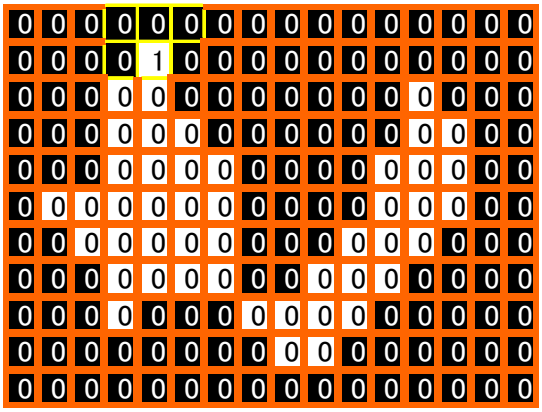


(a)ラベリング

1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

（b）ルックアップテーブル

図 5：ラベリングアルゴリズム　経過 1

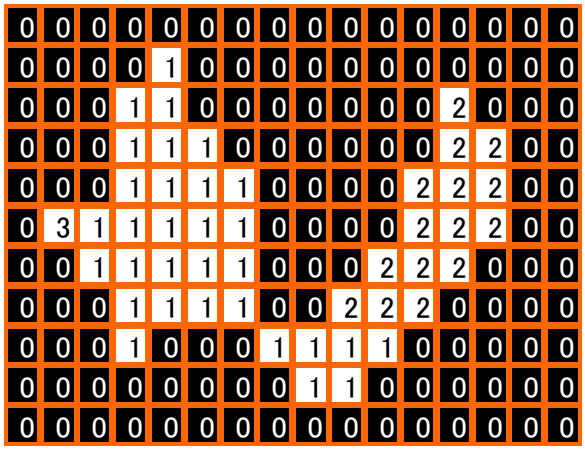


(a)ラベリング

1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

（b）ルックアップテーブル

図 6：ラベリングアルゴリズム　経過 2

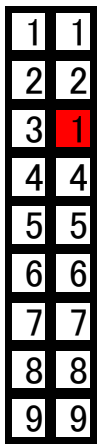


(a)ラベリング

1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

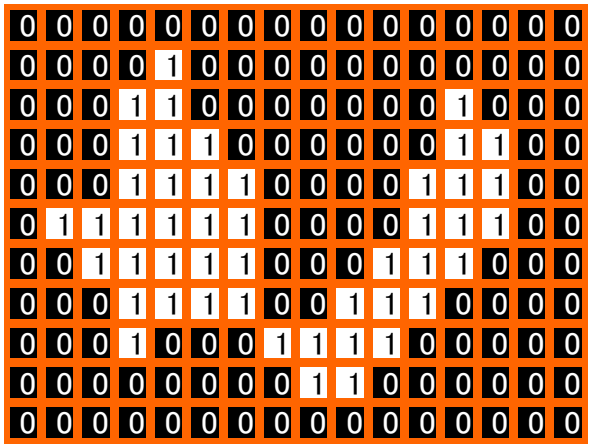
（b）ルックアップテーブル

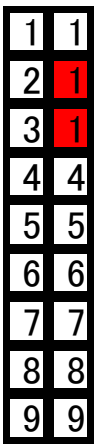
図 7：ラベリングアルゴリズム　経過 3



(a)ラベリング　　（b）ルックアップテーブル

図 8：ラベリングアルゴリズム　経過 4





(a)ラベリング　　（b）ルックアップテーブル

図 9：ラベリングアルゴリズム　経過 5

３　C 言語での欠損発見実験の検証

３．１　実験条件

C 言語での実験は、Harrier（CPU : Core2Duo2.66GHz、RAM : 3.0G）を使って行った。画像処理の対象は、横 1280×縦 960 の 8 ビットのビットマップファイルのガラス基板上のキズが写った画像である。

処理の手順は、対象となる画像にラプラシアンフィルタ、２値化、ラベリング行う。この時、TDI 処理を最初に行った場合と、行わない場合を比較する。比較は、肉眼での確認と処理速度とする。TDI は３２回、６４回、１２８回で比較する。　図 10 に実行手順のフローチャートを示す。

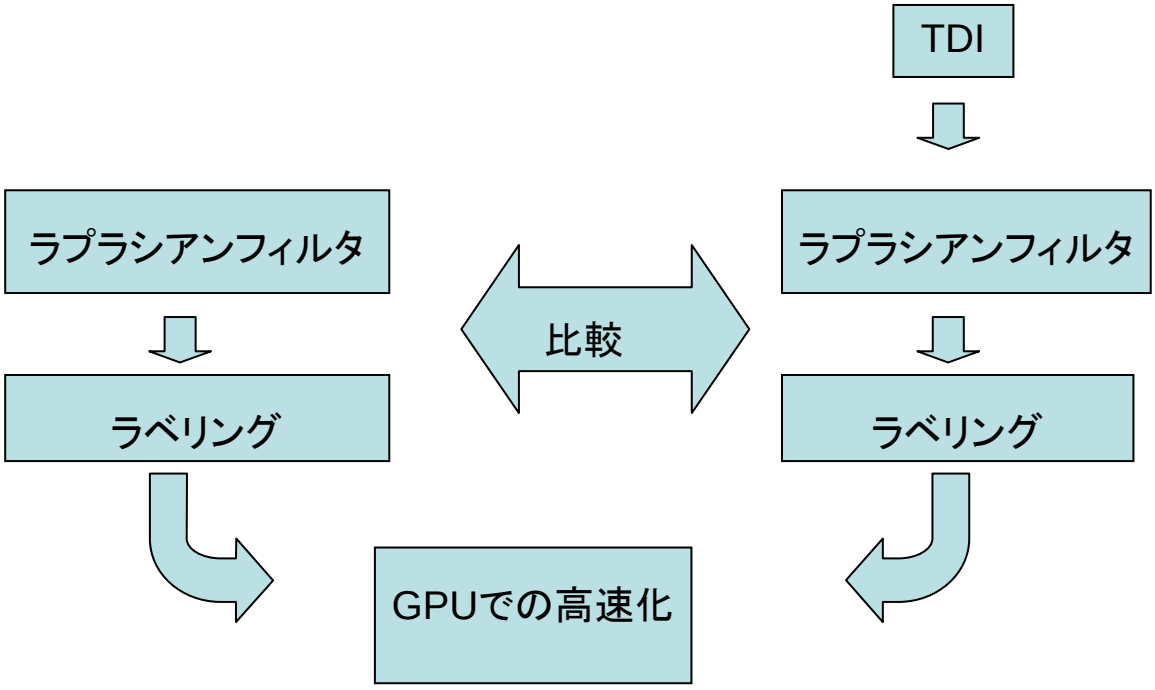


図 10：実行手順

３．２　TDI

TDI を３２回、６４回、１２８回で行った。入力画像と出力画像を図 11 に示す。
図 11 の赤い矢印で示している点が、欠損である。
また、表 1 に処理時間を示す。128 回＞64 回＞32 回の順でノイズが減少している。
処理時間は回数が増すごとに増えている。

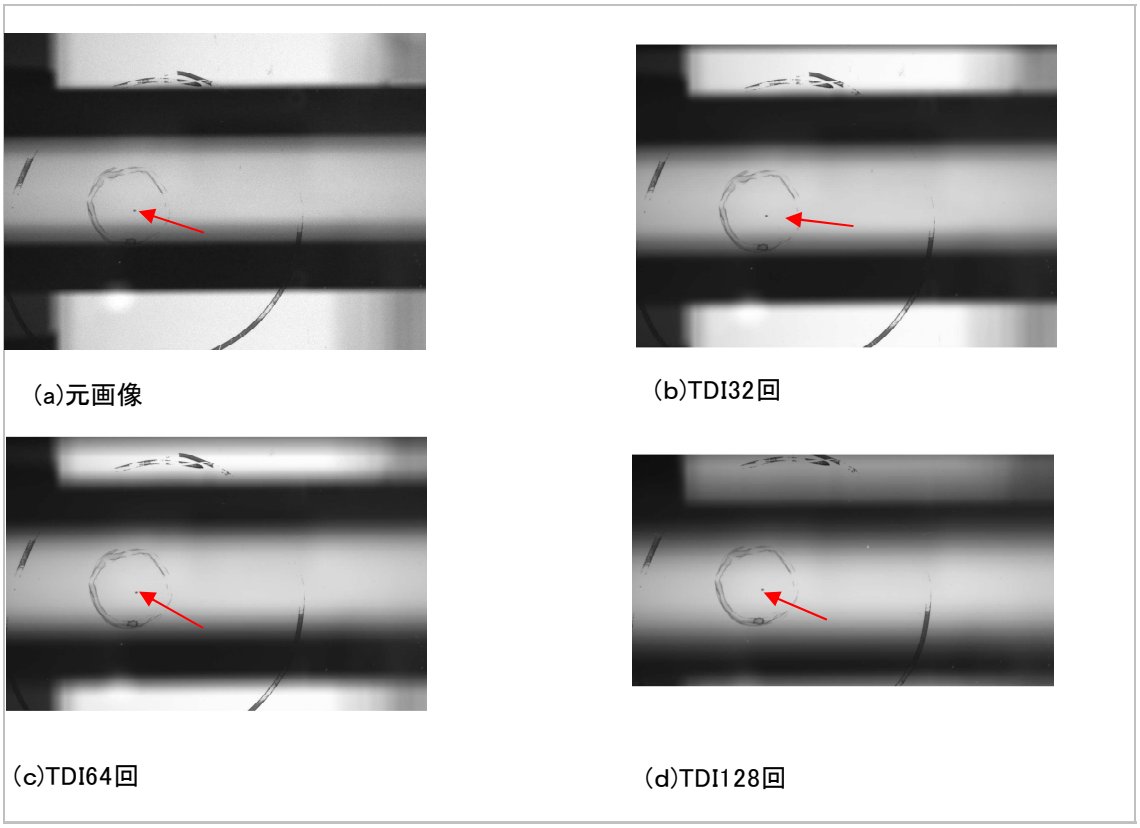


図 11：TDI　実行結果

表 1：TDI　処理時間

単位（秒）

TDI（回数）	32	64	128
処理時間	0.62	1.481	2.469

３．３　ラプラシアンフィルタ（８近傍）

元画像、TDI３２回、６４回、１２８回の画像にラプラシアンフィルタを掛けた後に、２値化した画像を図 12 に示す。図 12 では TDI の効果が分かりにくいので、拡大画像を図 13 に示す。また表 2 に処理時間を示す。　図 12 を確認すると、輪郭（エッジ）は TDI を行った場合、検出出来ている。　そして、図 12 の(a)を見ると分かるが、元画像にフィルタリングを行うと、ノイズを多く拾っており、輪郭（エッジ）部分がほとんど確認することが出来ない。図 13 を確認すると、TDI の効果がはっきりと確認出来る。１２８回＞６４回＞３２回の順で、ノイズが少なくなっていることが確認出来る。　処理速度では、ほとんど差が見られなかった。

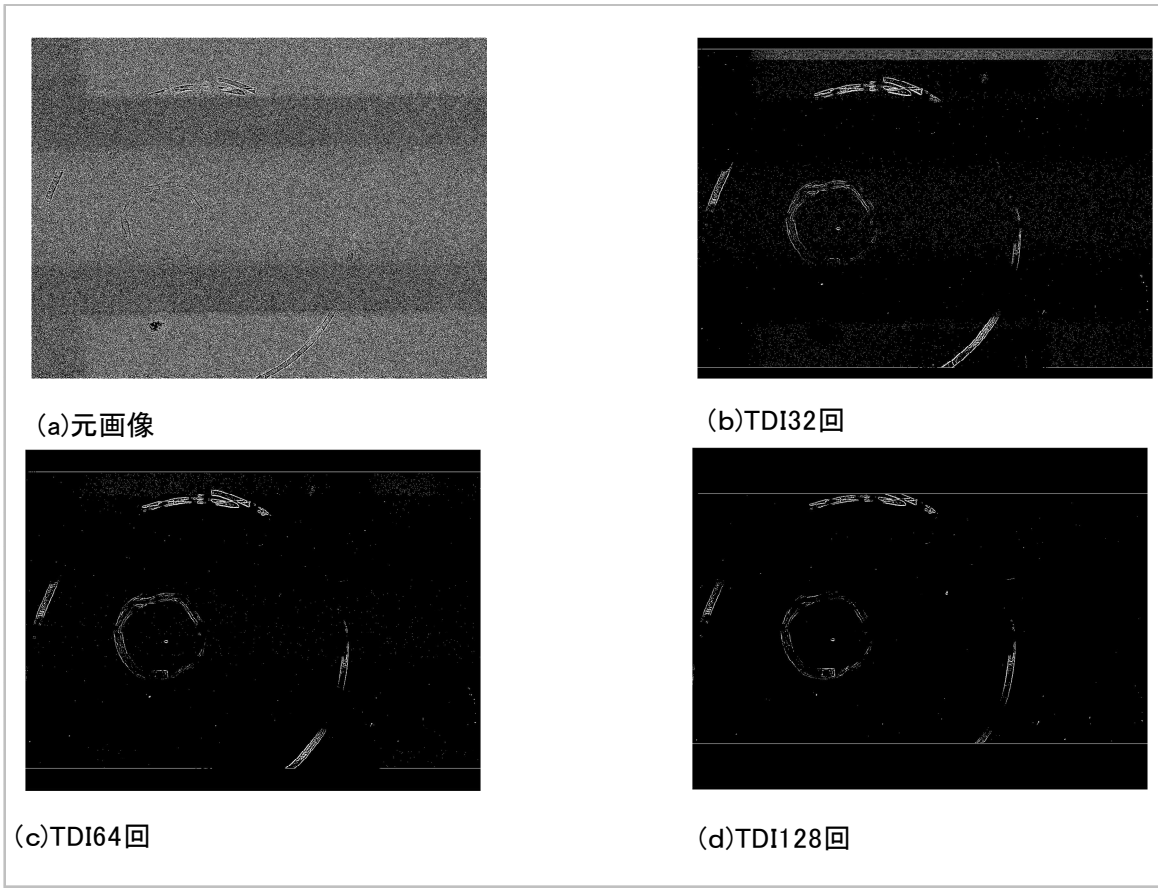


図 12：ラプラシアンフィルタ　実行結果

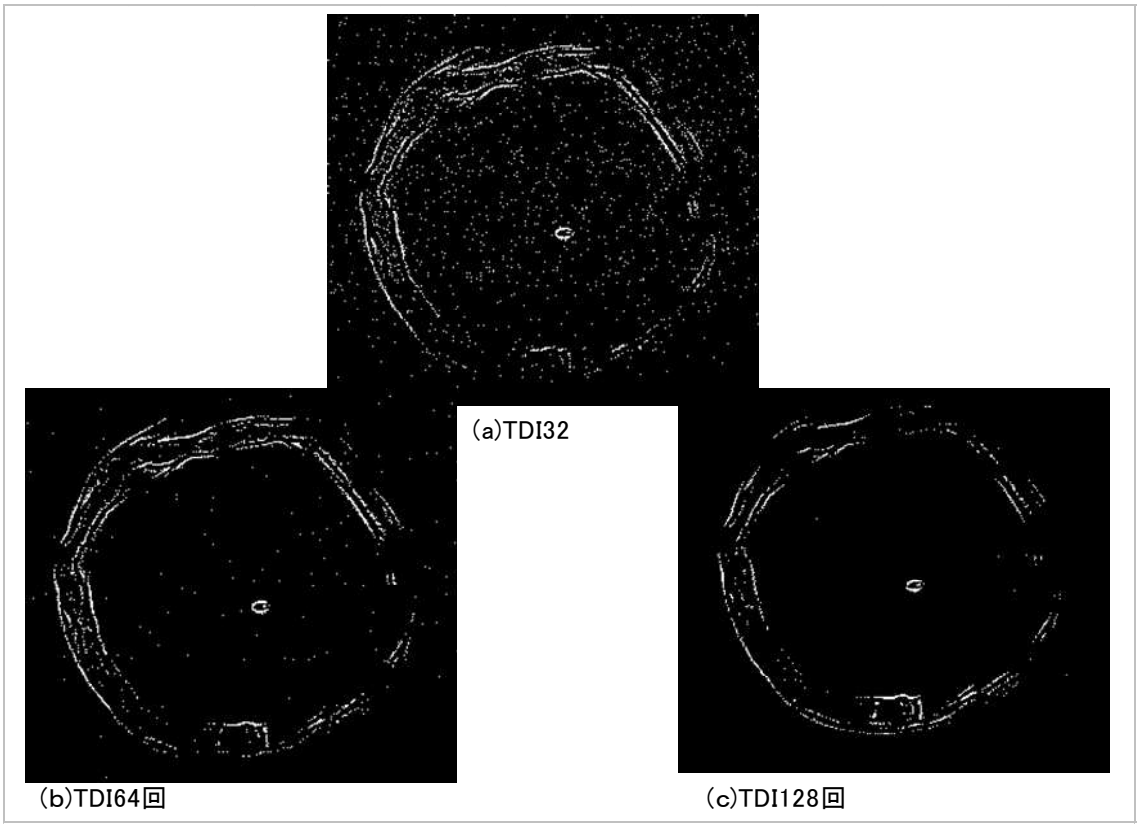


図 13：ラプラシアンフィルタ　拡大画像

表 2：ラプラシアンフィルタ　処理時間

単位（秒）

TDI(回数)	無し	32	64	128
処理時間	0.14	0.14	0.125	0.125

３．４　ラベリング

ここでは、ラプラシアンフィルタ処理を行った後に、２値化、ラベリングを行った。出力結果を図 14、拡大画像を図 15 に示す。比較対象は、元画像、TDI３２回、６４回、１２８回とする。また、処理時間を表 3 に示す。

図 15 の赤色の丸で囲まれている部分が欠損である。図 14 の（a）の元画像では検出出来ていないが、図 15 を見ると、TDI ３２回、６４回、１２８回共に、欠損部分を検出出来た。しかし、３２回では確認しづらい。

また、TDI３２回では、かなりノイズが残っており、６４回では、３２回と比べると、ノイズがかなり減少している。１２８回は、６４回に比べると、ノイズがより減っている。また、処理時間において、TDI３２回では、１２．２６倍、６４回では１４．６２倍、１２８回では２８．３６倍の速度向上であった。

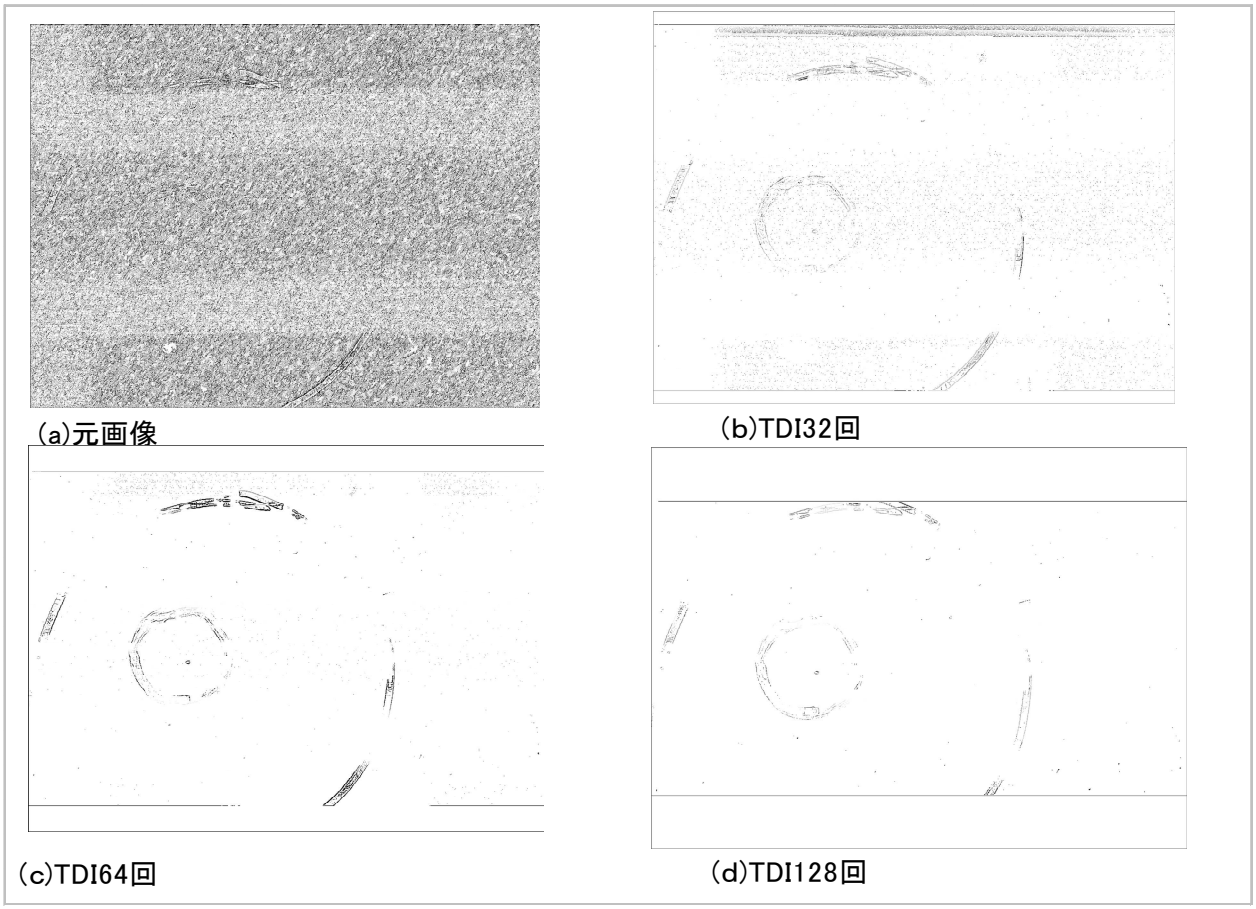


図 14：ラプラシアンフィルタ・ラベリング　実行結果

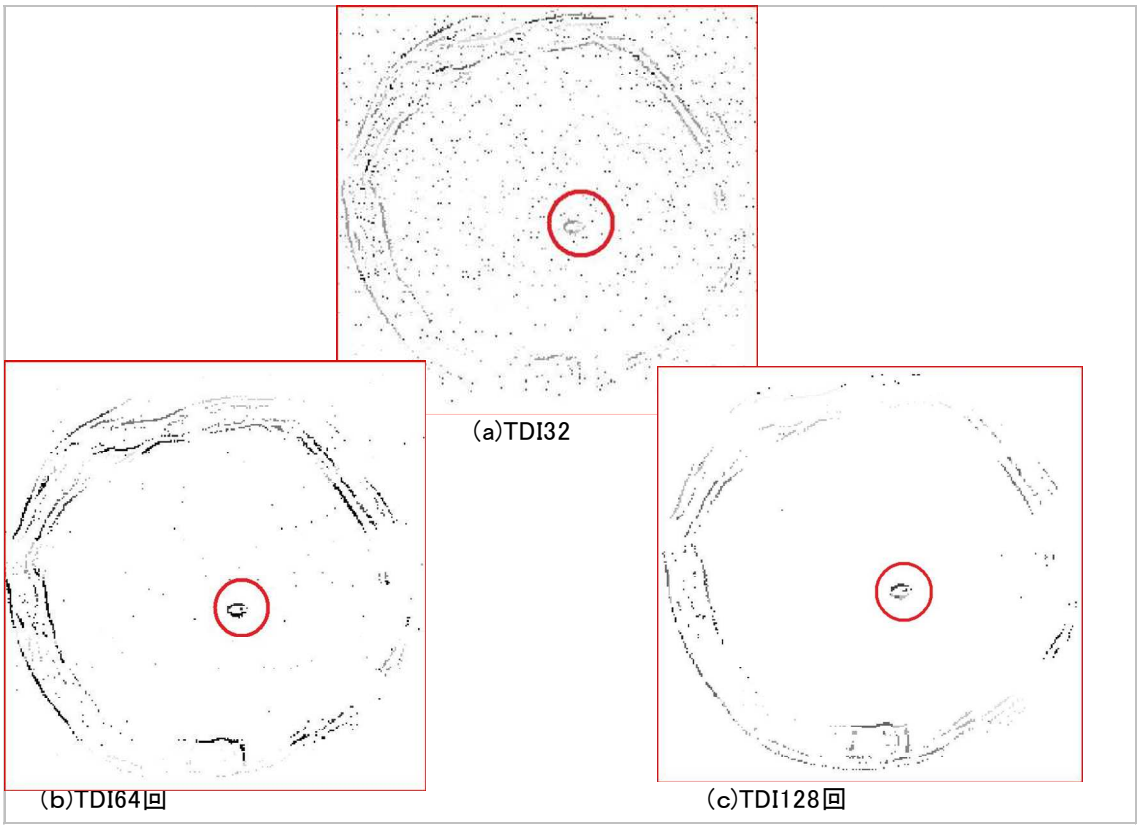


図 15：ラプラシアンフィルタ・ラベリング　拡大画像

表 3：ラプラシアンフィルタ・ラベリング　処理時間

単位（秒）

TDI(回数)	無し	32	64	128
処理時間	79.313	6.469	5.422	2.797

４　ＧＰＵを用いた画像処理の高速化

４．１　ＧＰＵと CUDA

ＧＰＵには､単純な計算を行う多数の演算装置が備わっており､その演算処理能力は CPU を上回る。GPU をグラフィック処理以外の汎用的な計算処理に使用することを、GPGPU という。その GPGPU を可能にしてくれるのが、CUDA である。CUDA は NVIDIA が提供する GPU 向けの統合開発環境であり、C 言語拡張して使うことが可能である。CUDA のハードウェアの中で一番小さな単位が､スカラプロセッサ（ＳＰ：Scalar Processor）で、この一つ上の単位にストリーミングプロセッサ（SM: Streaming Processor）となっている。ストリーミングプロセッサには、命令デコード機能が備わっており、デコードされた命令はスカラプロセッサに伝えられる。そのとき、搭載されているすべてのスカラプロセッサに、すべて同じ内容が伝えられる。伝えられたスカラプロセッサでは、同じプログラムが動作することにより、プログラムが複数のスカラプロセッサ上で並列に動くことになる。これにより大量のデータを並列に処理することが可能となる。 スカラプロセッサ上で動くプログラムの単位として、スレッドがあり、そのスレッドをまとめたものをブロックという。そのブロックをさらにまとめたものをグリッドと呼ぶ。このスレッドが敷き詰められて、一斉に処理を行う。１つのブロックは１つの SM 上で実行される。SP を 240 個搭載している場合、基本的に 30（240÷8）個の SM を搭載している。また、ブロックが SM の数を超えた場合は、適当な SM に複数のブロックが割り当てられ、タイムスライシングで平行実行される。また、デバイス（GPU）上で動作するプログラムを、カーネルプログラムといい、カーネルプログラムは、デバイス上で動作し、GPU が処理を行う。図 16 に GPU のハードウェア構成、図 17 にスレッドとブロックとグリッドの関係、図 18 に処理の流れを示す。

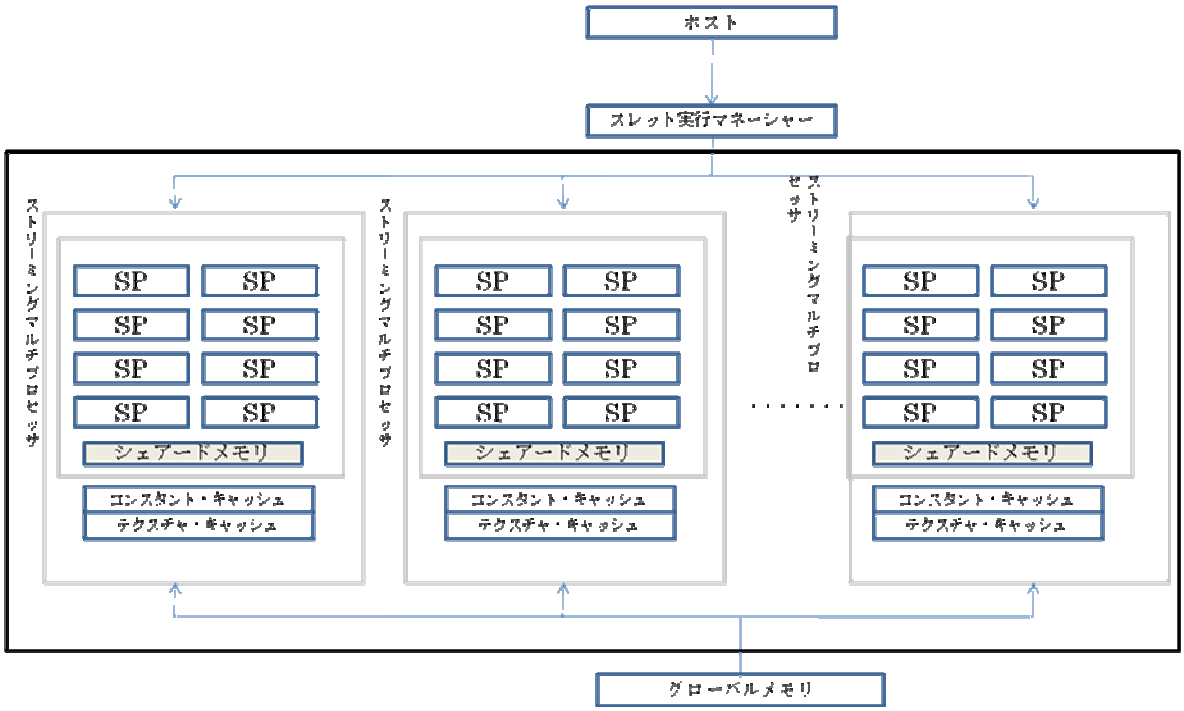


図 16：GPU のハードウェア構成

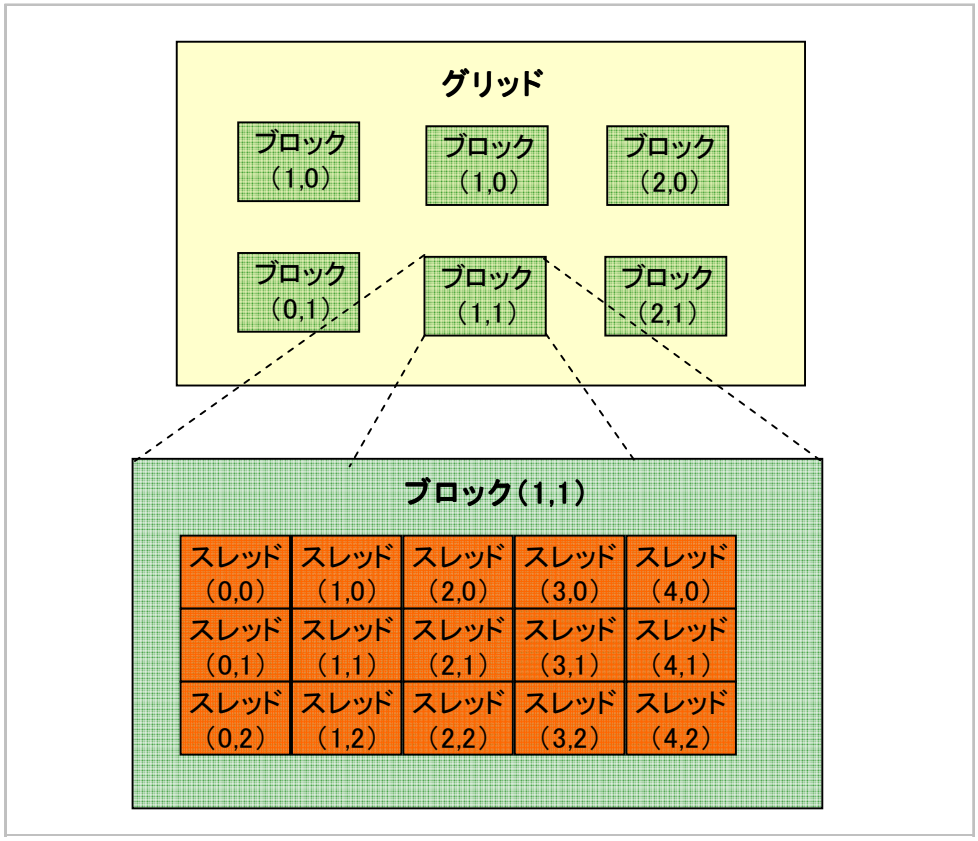


図 17：スレッドとブロックとグリッドの関係

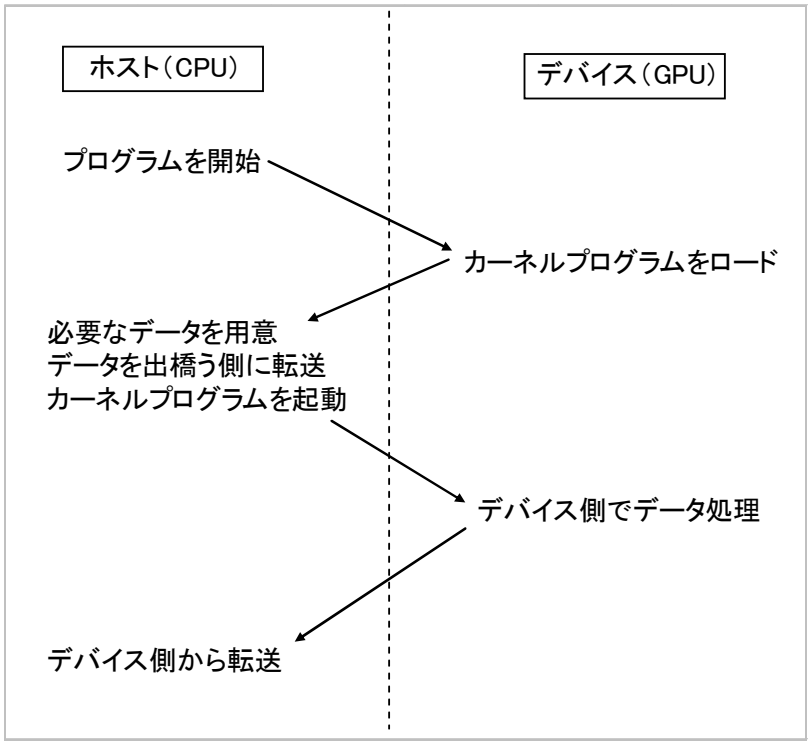


図 18：CUDA の処理の流れ

４．２　実験条件

GPU での実験は、Raputa(GeForce GTX480,SP：480 個、SM:15 基、RAM6.0GB、メモリバンド幅：177.4GB/sec）を使って行った。画像処理の対象は、横 1280×縦 960 の 8 ビットのビットマップファイルのガラス基板上のキズが写った画像である。比較は、肉眼での確認と処理時間とする。TDI は３２回、６４回、１２８回で比較する。今回の GPU は１つの SM に 32 個の SP（480÷15）があるので、最大で 32 スレッドが同時に実行される。

４．３　ラプラシアンフィルタ

今回はブロックを 40 個、スレッドを 32 個用意した。ビルトイン変数を用いた 1 次元配列へのアクセスイメージを図 19（40 ブロック、32 スレッドの場合）に示す。ビルトイン変数は、スレッドごとに違う値を持つ変数である。blockDim はスレッドの個数、blockIdx はブロックのインデックス番号、threadIdx はスレッドのインデックス番号を表している。画像に対しての並列化手法のイメージを図 20 に示す。まずスレッドを 1 列目の各要素に配置して、高さ分だけ処理を行う。各要素の計算がスレッドによって並列に実行されていく。処理後に２値化した画像を図 21、欠損部分の拡大画像を図 22 に示す。また表 4 に処理時間、表 5 に各ブロックとスレッド数での速度比較を示す。表 5 の比較では TDI128 回のデータで比較し、ブレがあるため 3 回の平均をとった。図 21、図 22 を見ると、CPU の場合と比較して、出力画像はエッジを抽出できた。少し処理部分を変えたので、単純に比較は行えないが、処理時間は表 4 を見ると、CPU の場合と比較して、処理速度が約 30 倍（元画像）向上した。表５を見ると、ブロック 40、スレッド 32 の場合がベストであった。

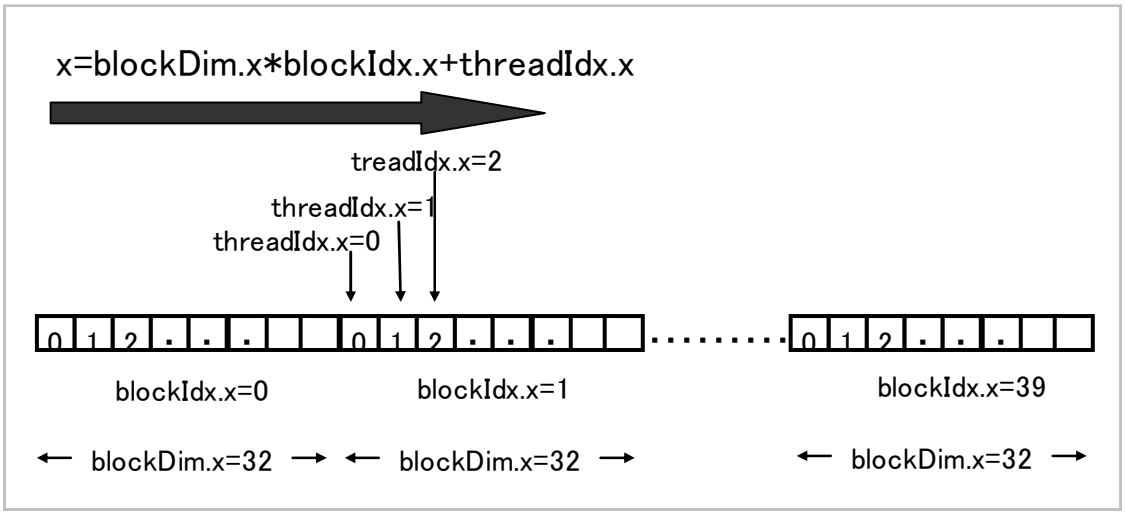


図 19：ビルトイン変数を用いた 1 次元配列へのアクセスイメージ（ブロック 40、スレッド 32）

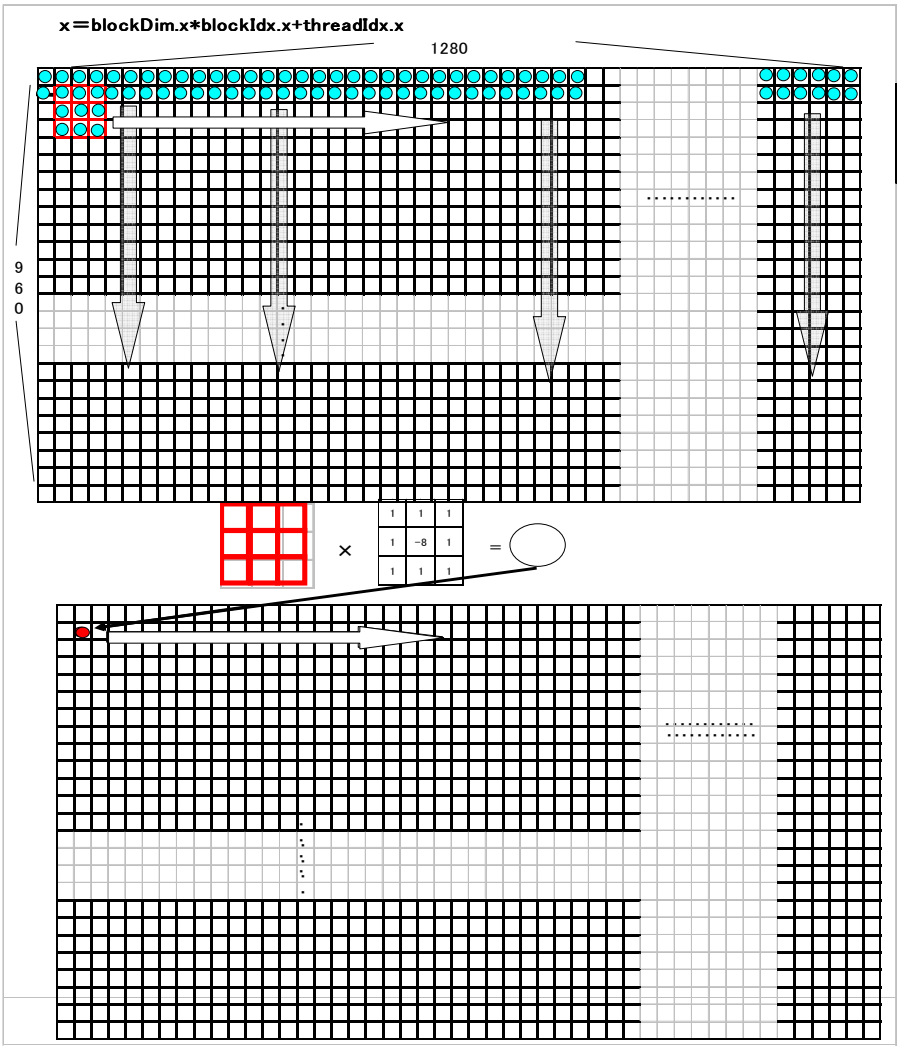


図 20：画像に対しての並列化手法のイメージ

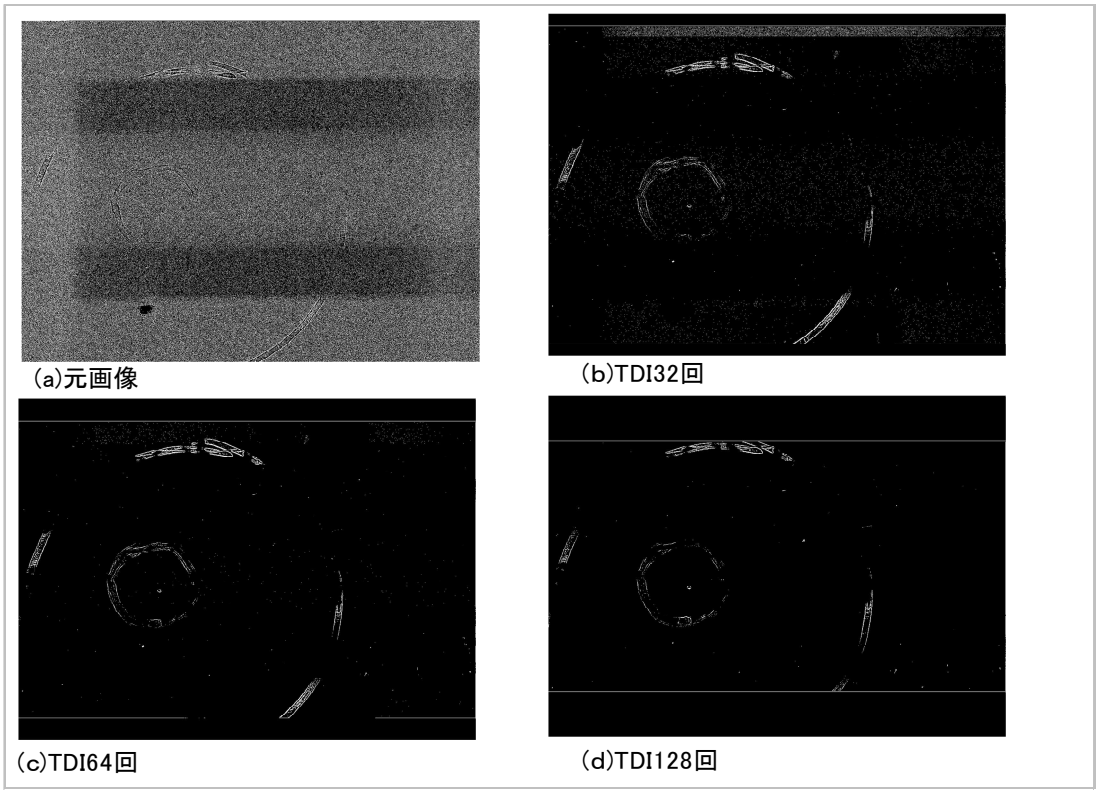


図 21：ラプラシアンフィルタ（GPU）　実行結果

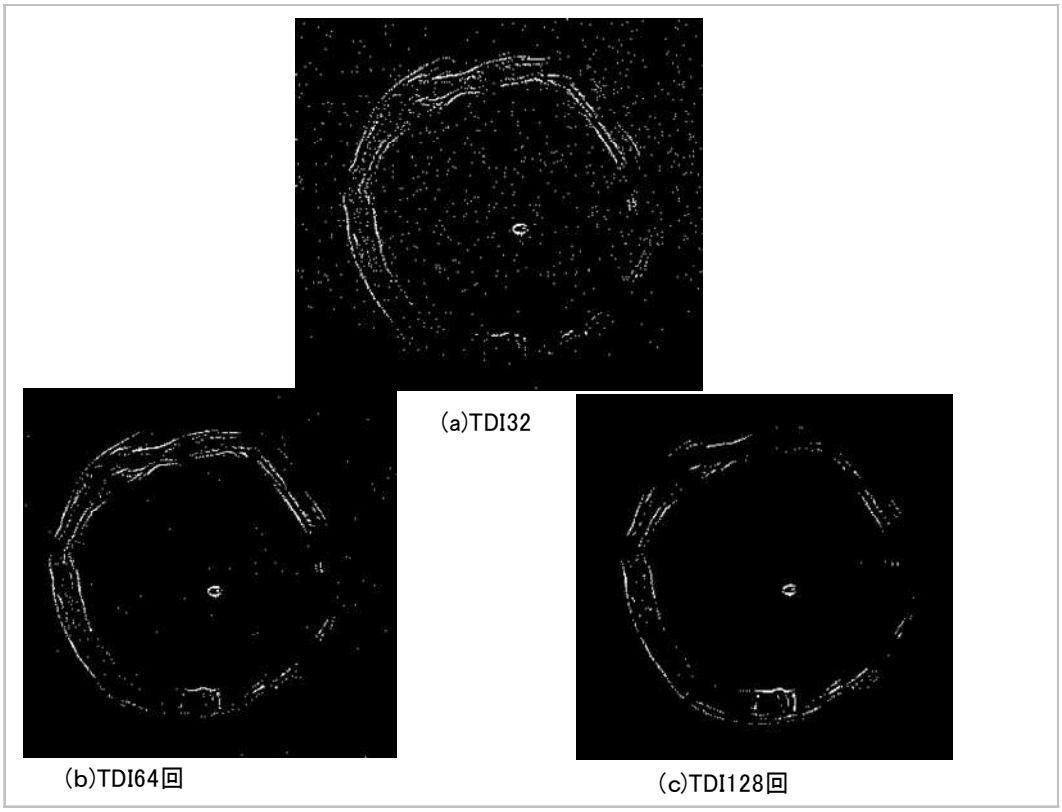


図 22：ラプラシアンフィルタ（GPU）拡大画像

表 4：ラプラシアンフィルタ（GPU）処理時間　　単位（秒）

TDI(回数)	無し	32	64	128
CPU	0.14	0.14	0.125	0.125
GPU	0.0046	0.0045	0.0044	0.0042

表 5：各ブロック数とスレッド数での速度比較　　単位（ミリ秒）

ブロック、スレッド	160,8	80,16	40,32	20,64	10,128
処理時間	6.2296	4.0741	4.0099	4.1167	4.2256

5　考察

CPU での実験では、ラプラシアンフィルタ、ラベリングの両方で TDI の効果を確認出来た。出力画像での確認では、ラプラシアンフィルタとラベリングの両方で、TDI を行わないで処理を行うと、欠損部分を確認出来なかったが、TDI を行った場合は欠損部分を確認出来た。また、TDI の回数を増やすほどノイズが少なくなっていることも確認出来た。処理時間においては、ラプラシアンフィルタの場合は元々の処理時間が短いので、あまり効果が見られなかったが、ラベリングでは TDI32 回で、12．26 倍、6 4 回では 14．62 倍、128 回では 28．36 倍の速度向上が見られた。TDI 処理時間を足しても 128 回の場合、約 15 倍の速度向上であった。これらのことから、TDI は回数を増やすほど効果が上がり、適当な回数は、128 回が良いと思われる。

GPU での高速化では、ラプラシアンフィルタのみ実装出来た。出力画像は、CPU の場合と比較すると、輪郭は抽出でき、ノイズも少なかった。少し処理を変えたので、単純に速度比較は出来ないが、全体で約 30 倍高速化を達成できた。今回は、1 次元配列への x 方向だけにスレッド、ブロックを配置したが、y 方向にもスレッドを配置すると、さらに高速化を望めると思われる。各ブロックとスレッドでの速度比較では、SM はワープ（32 スレッド)、ハーフワープ（16 スレッド）と呼ばれる単位で処理されるので、(40、32)、(80、16)、(20,64）あたりの処理時間が短くなったと、思われる。(160、8）が遅いのは、スレッドが 8 となっており、ワープ、ハーフワープになっていなからであると、考えられる。

6　おわりに

本研究では、GPU を用いた液晶用ガラスの欠損検出画像処理の高速化を目的とし、まず CPU で「TDI」、「ラプラシアンフィルタ」、「ラベリング」を行った。CPU では TDI の効果を確認でき、出力画像では TDI によりノイズがかなり減り、欠損を肉眼で確認する事が可能になった。処理時間においては、「ラプラシアンフィルタ」、「ラベリング」の時に、最大で約 28 倍の速度向上が得られた。

GPU では、「ラプラシアンフィルタ」のみの実装を行った。出力画像が CPU の場合と少し異なるので、単純に処理時間の比較は行えないが、CPU の場合と比較して全体で約 30 倍、処理速度が向上した。「ラベリング」処理は実装を行えなかったが、処理が重いので、並列化の効果が期待出来る。今後の課題としては、GPU での並列化を行って、「ラプラシアンフィルタ」、「ラベリング」処理の正しい出力画像と速度向上を得ることである。

謝辞

本研究の機会を与えて下さり、貴重な助言、ご指導を頂きました山崎勝弘教授に深く感謝いたします。また、共同研究において、この機会と研究材料である画像データを与えてくださった株式会社ケー・デー・イーの西田氏、三宅氏に感謝いたします。 最後に、本研究に関して様々な相談に乗って頂き、貴重なご意見を頂きました高性能計算研究室の皆様に深く感謝いたします。

参考文献

【１】 岡田賢治：CUDA 高速 GPU プログラミング入門、秀和システム、2010

【２】 粂井康孝：猫でもわかる C 言語プログラミング、ソフトバンククリエイティブ、２００４

【３】 内村 圭一、上瀧 剛：実践 画像処理入門、培風館、2007

【４】 大山 佳宣：OpenMP を用いた画像処理プログラムの並列化、立命館大学電子情報デザイン学科卒業論文、2010

【５】 NVIDIA： CUDA テクニカルトレーニング Vol I ：CUDA プログラミング入門、http://www.nvidia.co.jp/docs/IO/59373/VolumeI.pdf

【６】 千村亮介：ラベリングとマハラノビス距離による画像判別の実現、立命館大学電子情報デザイン学科卒業論文、2006

【７】 山内寛紀： 電子情報デザイン実験Ⅲ 実験指導書、2009、電子情報デザイン実験ⅢB 画像・映像処理 p 13-20 、立命館大学 理工学部 電子情報：2009 年度

【８】 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、 立命館大学電子情報デザイン学科卒業論文、2006

【９】 松崎裕樹、山崎勝弘、平岡邦廣、西田洋隆、三宅淳司：マハラノビス距離を用いたガラス検査用画像判別の実現、 平成 18 年度情報処理学会関西支部 支部大会講演論文集 C-09, p.187-190, 2006

【１０】青木尊之、額田彰：はじめての CUDA プログラミング、工学社、2009