Vol.2010-CG-139 No.5
2010/7/16

類似画像検索における GPU の活用

杉浦司[†]　田中敏光[†]　佐川雄二[†]　手島裕詞[††]

一般に類似画像検索では，画像に様々な処理を施して抽出した特徴を比較している．しかし，画像から特徴量を抽出する処理には多くのコストがかかるため，対象とする画像が多数になると検索時間がかかりすぎる問題がある．本研究では，特徴抽出に GPU を活用することで類似画像検索の処理時間の短縮を図る．モルフォロジー演算を用いる特徴抽出処理を GPU（GeForce GTX285）に実装した場合，CPU（Core i7 920）で実行した場合に比べて類似画像検索全体で約 3 倍の高速化が見込める．

Similar Image Retrieval accelerated by using GPUs

Tsukasa Sugiura[†]　Toshimitsu Tanaka[†]　Yuji Sagawa[†]
Yuji Teshima[††]

The similar image retrieval is one of the technologies that efficiently retrieve desired images from a lot of images. Generally, in the similar image retrieval, various processing is applied to images, features of the images are extracted, and finally the features are compared. However, since the feature extraction is very costly, it increases total retrieval time. To accelerate the feature extraction process, we implement the process on GPU. In an experiment of a program using the morphological operations for similar image retrieval, it is expected that GPU(GeForce GTX285) runs the program over three times faster than CPU(Core i7 920).

1. はじめに

近年，安価なデジタルカメラや大容量の記憶装置の普及に伴い，ユーザーが保存する画像は飛躍的に増大している．これに伴い，以前に撮影した画像を見つけられなかったり，見つけるまでに長い時間がかかったりする問題が起こっている．このため，効率的な画像検索技術が求められている．

画像検索技術の一つである画像内容から類似画像を検索する手法では，一般に，画像に様々な処理を施すことで画像内容を表す特徴量を取り出し，比較している．しかし，この特徴抽出処理には多大なコストがかかり検索時間増大の原因になっている．このため，参考文献[1]の研究では，処理を専用プロセッサに実装して高速化している．このように，ハードウェアで処理することは非常に有効な高速化手段であるが，専用プロセッサを使っていたのでは，価格の高さや入手性の低さが問題になる．一般ユーザーが容易に利用できるものにするには，広く普及しているハードウェアを用いる必要がある．

コンピュータグラフィックス（以下 CG）の処理を高速に実行するために開発されたグラフィックスプロセッシングユニット（以下 GPU）は，現在ゲームのアクセラレータとして広く普及している．さらに，Intel 社も AMD 社も CPU のパッケージ内に GPU を統合する動きを進めているので，近々，PC を購入するとある程度の処理能力を持った GPU が付属するようになる．最近では，GPU の演算性能の高さに着目して，GPU を物理演算や動画像コーディングなどの処理に利用しているが，このような背景から，GPU の利用用途のよりいっそうの拡大が求められている．

本研究では，GPU を活用して類似画像検索の処理時間を短縮する．

2. 実装環境

2.1 GPU（Graphics Processing Unit）

GPU は，CG を高速に処理するために開発された演算ユニットである．CPU と GPU の構成図を図 1 に示す[2]．GPU は単純な演算ユニットを数多く搭載しており，これらを並列に動作させることで高速処理を実現する．主にグラフィックスボードと呼ばれる拡張カード上に実装された形で提供されており，個人所有の PC にも簡単に装着できる．

GPU が登場した当初は，CG で必要とする処理を専用の回路で実現していたが，技術の発展に伴い，汎用計算を処理できる構造を持つようになった．近年では，CPU を

[†] 名城大学大学院
Meijo University
[††] 静岡理工科大学
Shizuoka Institute of Science and Technology

遥かに超える演算性能を持つようになり，より汎用的な処理に対応できる構造になっている．GPU の演算性能を図 2 に示す[2]．このような特徴が注目され，GPU を CG 以外の用途に用いる GPGPU（General-purpose computing on graphics processing units）と呼ばれる動きが見られ，様々な用途に活用され始めている．

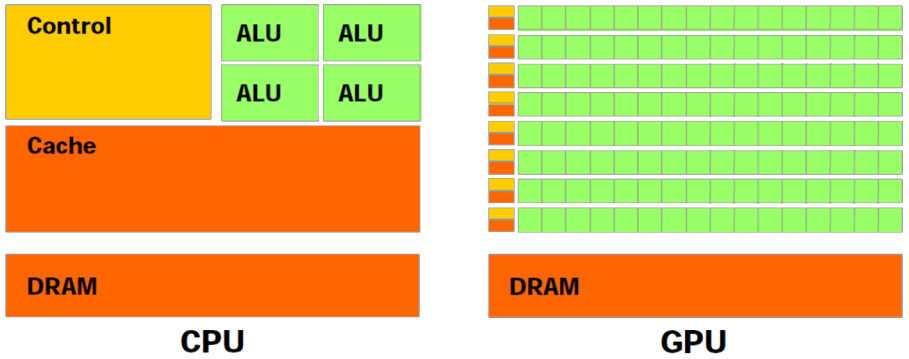


図 1　CPU と GPU のブロックダイアグラム

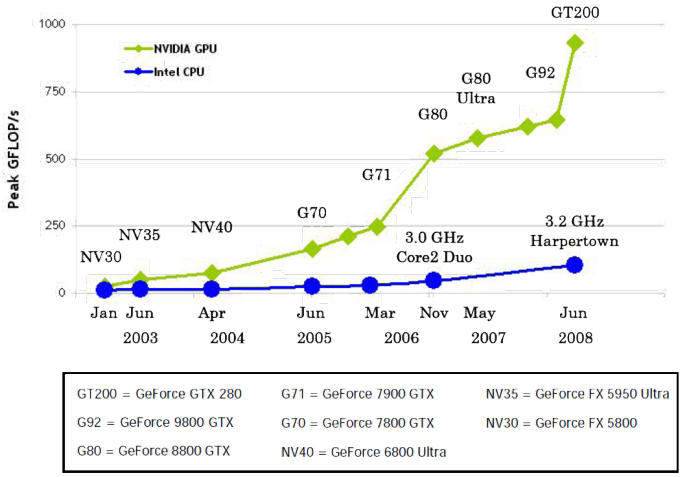


図 2　CPU と GPU の演算性能の比較

2.2　CUDA（Compute Unified Device Architecture）

従来は，Microsoft 社の HLSL や OpenGL ARB WG の GLSL などのプログラミング言語を用いて，GPU プログラミングを行っていた．しかし，これらの言語は，CG の演算を実行するために設計されているため，汎用的な計算を行うには扱いづらかった．

このため，最近では，GPU を汎用計算に使うために設計されたプログラミング開発環境が，各ベンダから提供されるようになった．その一つに NVIDIA 社が提供する CUDA がある．CUDA は一般的な C/C++言語を拡張した仕様になっているため，GPU を比較的容易にプログラミングできる．また，従来の開発環境に比べ，GPU の扱いが抽象化されているため，頻繁に行われている GPU のアーキテクチャの変更に柔軟に対応できる．

これらの理由から，本研究では CUDA を用いて類似画像検索を GPU 上に実装することで，処理の高速化を図る．表 1 に本報告の各検証実験で使用した環境を示す．

表 1　実験環境

CPU	Intel Core i7 920 2.66GHz
GPU	NVIDIA GeForce GTX285
CUDA	CUDA 2.3
OS	Microsoft Windows Vista Business 32bit
Driver	GeForce/ION Driver Release 195.62

3.　GPU を用いた画像処理

GPU の高い演算性能を引き出すには，GPU 独特のアーキテクチャを理解し，最適化していく必要がある．最近の GPU には，200 個以上の演算ユニットが搭載されており，1 つのデータに対して多くの演算ユニットを用い並列計算することで，高速処理を実現している．画像処理では，画素単位の計算を他の計算結果に依存すること無く独立して行える場合が多いため，これを各演算ユニットに分散して割り当てることができ，GPU で高速に処理できる．例えば，フィルタ処理や表色系変換処理といった画像処理は GPU で高速に行える可能性が高い．

ただし，GPU は CPU 側のホストシステムとは独立したデバイスとして扱うため，処理対象となるデータは予めデバイス側のメモリに転送しておく必要がある．この関係を図 3 に示す．ホストとデバイスの間のデータ転送には PCI-Express を通す必要があるが，PCI-Express の転送速度がボトルネックになり時間がかかる．そのため，画像を何度もやり取りするような実装では処理は速くならない．また，画像全体からサンプルを選んで処理するような，局所性が無いアルゴリズムでは，メモリアクセスが競

合したり，読み出しに時間がかかったりするため，かえって処理に時間がかかる．GPU は CPU に比べ分岐粒度が荒いため，条件分岐処理が苦手である．このため，GPU プログラミングでは，データの転送やメモリアクセス，条件分岐処理に注意を払う必要がある．

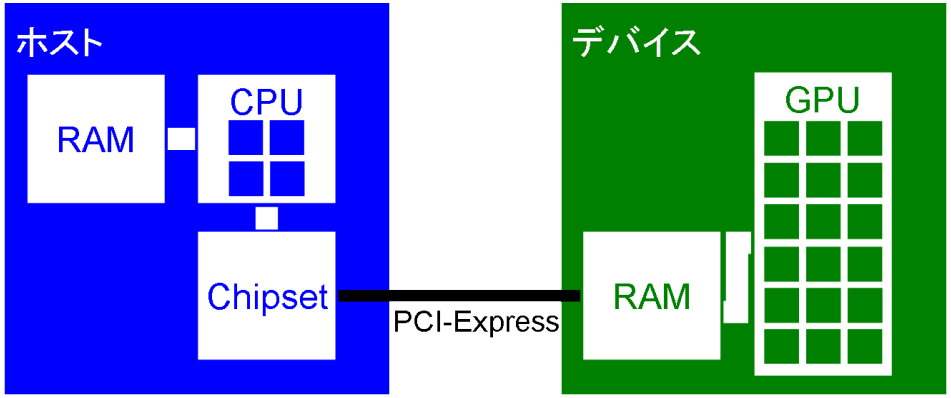


図 3　ホストとデバイスの構成図

4. 類似画像検索における GPU の活用方針

4.1 実装方針

汎用計算に適した開発環境が整ってきたとはいえ，GPU でのプログラミングは CPU より手間がかかる．そこで，これまでに性能が確認されている類似画像検索アルゴリズムの中から，GPU で高速に動作することが確認できた画像処理を多く使うものを選び，GPU 向けに改良することから始める．

本研究では，類似画像検索アルゴリズムに，筆者の提案したモルフォロジー演算を使う手法[3]を用いる．

4.2 モルフォロジー演算を用いた類似画像検索

モルフォロジー演算を用いた類似画像検索の処理の流れを図 4 に示す．まず，入力された画像から特徴を抽出する．そして，入力画像と比較対象の画像の間で特徴量の類似度を計算する．最終的に類似度を比較することにより，どの画像が入力画像に似ているかを判別する．

図 4　類似画像検索の処理の流れ(簡略図)

一連の検索処理の主要部分である特徴抽出処理は，大きく分けて以下の処理で構成されている．

(1) **モルフォロジー演算を用いた色変化の抽出**

入力画像に対して特定方向に強い反応を示す構造関数を用いてモルフォロジー演算を行う．そして，演算前の画像との差分をとることで，演算による色変化を抽出する．ここでは，カラー画像の各チャンネルに対して 4 種類の構造関数を用いこの処理を行う．

(2) **画像領域の分割**

上記の処理で得られた色変化抽出画像に対して 64 分割になるまで再帰的に 4 分割していく．分割数は類似画像検索実験で最も評価が高くなった値を用いている．

(3) **各分割領域からのカラーヒストグラム計算**

分割された各画像領域からカラーヒストグラムを計算する．さらにヒストグラムの正規化を行う．

4.3 処理コストの分析

モルフォロジー演算を用いた類似画像検索を CPU のみで実行すると，表 1 に示した実験環境で 500 万画素の画像を処理した場合，1 枚あたりの処理に約 4 秒の処理時間がかかる．この処理時間の内訳を図 5 に示す．

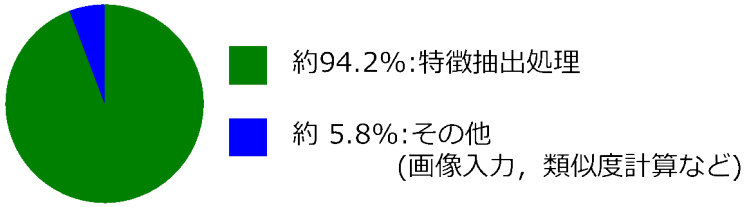


図 5　類似画像検索の処理時間内訳

検索時間の約 94.2%を特徴抽出処理に費やしており，この部分を高速化することが重要であることがわかる．

次に，特徴抽出処理の処理時間の内訳を図 6 に示す．

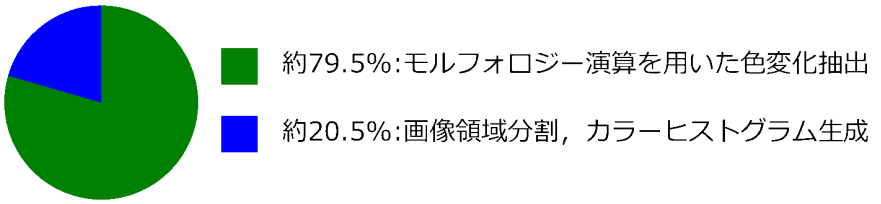


図 6　特徴抽出処理の処理時間内訳

処理時間のかかるモルフォロジー演算を 1 枚の画像につき 12 回行っているため，特徴抽出処理全体の約 79.5%がモルフォロジー演算に費やされている．そこで，まずモルフォロジー演算を GPU 上に実装して高速化する．

4.4 GPU の活用方針

以上のことから，特徴抽出処理を GPU 上に実装し，類似画像検索の高速化を図る．

但し，データ転送量を最小限に抑え，GPU を効率的に利用するために，モルフォロジー演算部分のみだけではなく，特徴抽出処理の一連の処理を全て GPU 上に実装する．これは，処理した画像ではなく，抽出した特徴量をホスト側に返すことで，転送するデータ量を大きく削減できるからである．

また，ホストとデバイス間のデータ転送を GPU 上での処理と並行して行うことで，データ転送時間を隠蔽する．この様子を図 7 に示す．

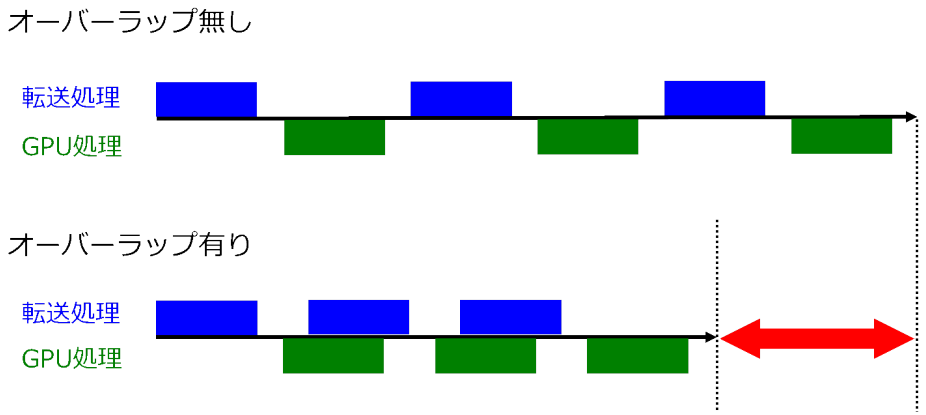


図 7　データ転送処理と GPU 処理のオーバーラップ

5. 実験

現在実装が完了している特徴抽出部分について，その性能が十分なものであるか処理速度を計測し，検証する．各実験では，最小の画像サイズとして携帯電話のカメラで撮影可能な 500 万画素(2594pix×1944pix)，最大サイズとして一般的なデジタルカメラで撮影可能な 1000 万画素(3888pix×2592pix)，その間の 750 万画素(3036pix×2434pix)の 3 種類の画像サイズを用いて検証を行った．

5.1 モルフォロジー演算を用いた色変化抽出

4.2 の(1)で示したモルフォロジー演算を用いた色変化抽出処理について，処理時間を計測した．この計測結果を図 8 に示す．実験では，画像サイズごとに 10 枚の画像に対して処理を行った．また，この処理は特徴抽出の最初の処理であるため，画像の転送処理から処理終了までの時間を計測している．

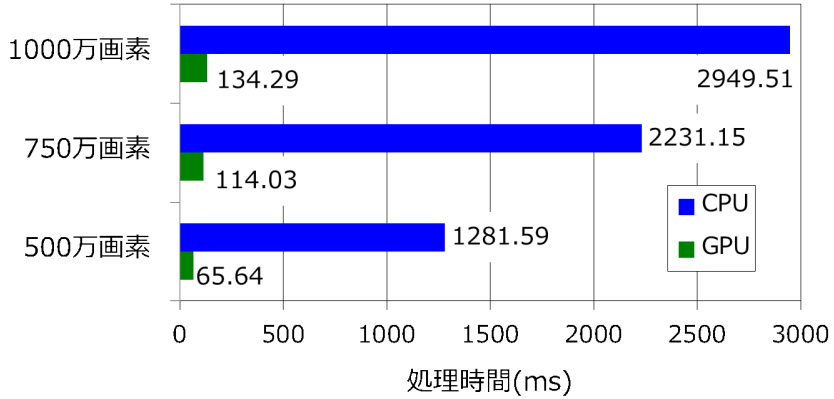


図 8　モルフォロジー演算を用いた色変化抽出の処理時間

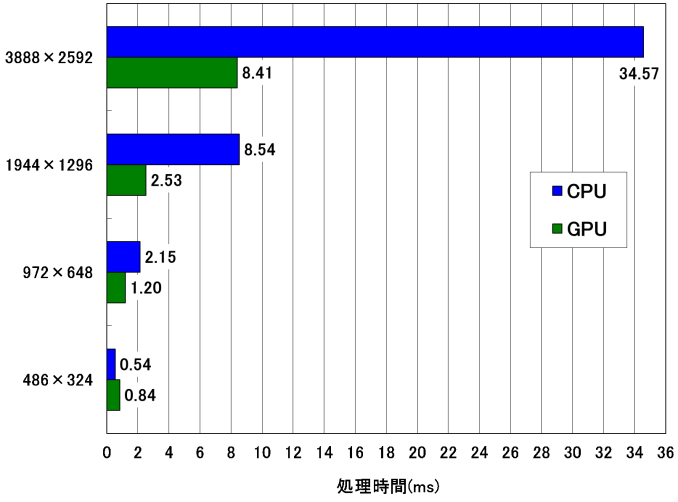
この計測結果から，GPU 上に実装すると CPU に比べ約 20～22 倍高速に動作することが確認できた．また，画像サイズが大きくなるにつれ，高速化倍率が高くなることが示された．昨今のデジタルカメラでは，1000 万画素以上の解像度があたりまえになっているため，実際の利用条件では，いっそうの処理速度改善が予想される．また，今回はアルゴリズムを素直に GPU に移植しただけなので，高速なオンチップメモリを利用する，メモリアクセスを最適化する，等の改良を加えることで，より高速に動作することが期待できる．

5.2 各分割領域からのカラーヒストグラム計算

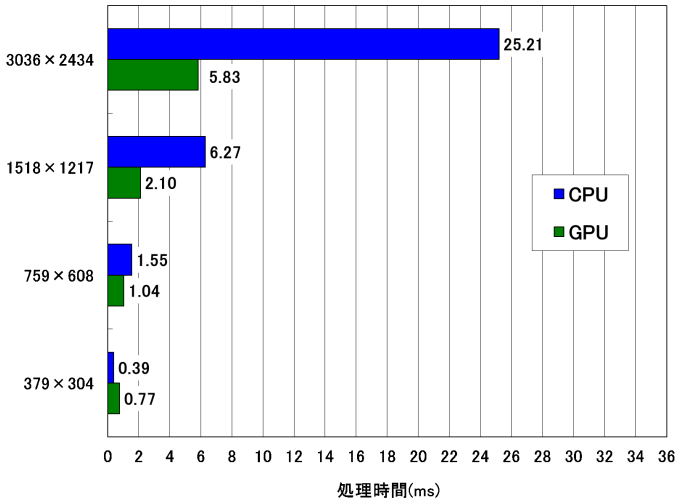
4.2 の(3)で示した，再帰的に 4 分割することで 64 分割した画像に対して，それぞれの分割階層ごとに部分画像のカラーヒストグラムを計算する処理を行い，処理時間を計測する．

まず，各分割サイズの画像からカラーヒストグラムを計算する平均処理時間を計測した．この結果を図 9 に示す．ヒストグラム計算には参考文献[4]のサンプルコードを画像向けに調整したものを利用した．この結果から，分割画像のサイズが大きい場合は CPU よりも高速に処理できるが，画像サイズが小さくなると GPU での並列計算のメリットより処理を分配するオーバーヘッドが大きくなって，CPU の方が若干速く処理できる結果になった．

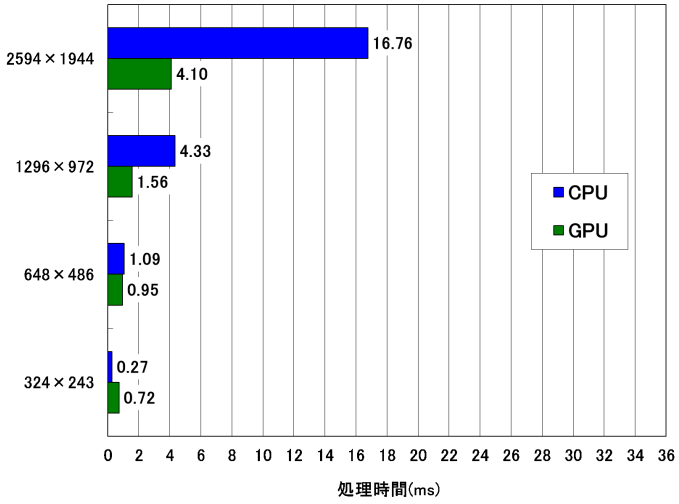
次に実際の特徴抽出での処理時間を検証する．実際の特徴抽出では，分割領域数分の計算を行うため，それらを足しあわせた処理時間を図 10 に示す．この結果から，1000 万画素の場合は CPU に比べて約 1.5 倍，750 万画素の場合は約 1.3 倍の速さで処理できることが確認できた．500 万画素の画像では大きな差ではないものの，CPU の方が若干速く処理できる結果となった．しかし，画像をホスト側に書き戻すよりも特徴量を書き戻す方が低コストのため，GPU で処理した方が有利である．



(a) 1000 万画素



(b) 750 万画素



(c) 500 万画素

図 9 各分割サイズからのカラーヒストグラム計算の処理時間

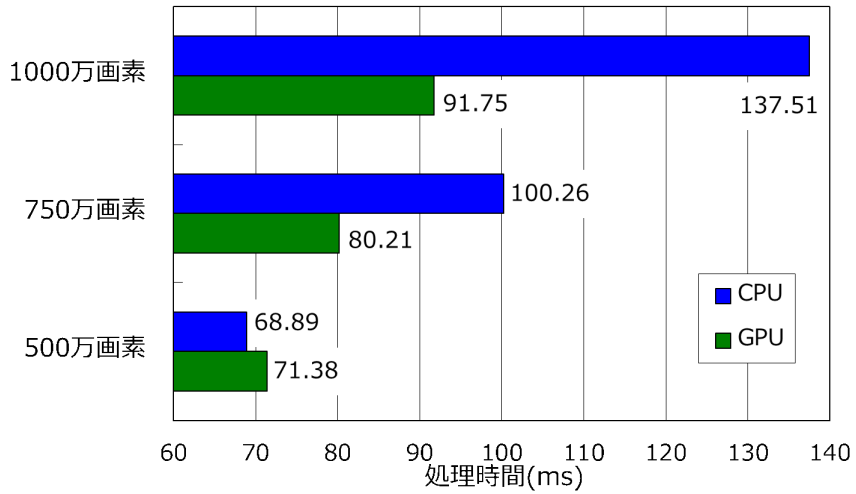


図 10　各画像サイズからのヒストグラム計算の処理時間

5.3 考察

5.1 の結果から，モルフォロジー演算を用いた色変化抽出処理では，少なくとも 20 倍の高速化が見込める．5.2 の結果から，各分割領域からのカラーヒストグラム計算では少なくとも CPU と同程度の処理速度が見込める．また，画像領域の分割処理についても CUDA の API を用いることで CPU と同程度の速度で行えることが確認できている．これらの結果をまとめると，

特徴抽出処理の割合（94.2%）×モルフォロジー演算の割合（79.5%）
×モルフォロジー演算の削減率（1－1/20）＝71.1%

が削減が期待される計算コストなので，類似画像検索全体では，CPU に比べて処理時間で約 70%の短縮，率では約 3 倍の高速化が期待できる．

6. おわりに

本報告では，類似画像検索の高速化に GPU を活用することを提案し，その実装方針を示した．また，現在実装が修了している部分の処理速度を検証した．実験結果からは，GPU を活用することで類似画像検索を 3 倍程度は高速化できる目途が立った．

これまでに特徴抽出処理における各処理の実装が修了したので，今後は類似画像検索全体としての実装を行い，処理速度を検証する．また，実装したアルゴリズムは，GPU 向けに設計したものではないため， GPU の演算性能を十分に引き出しているわけではない．そこで，プロファイラ等を用いてボトルネックとなっている部分を洗い出すし，GPU のアーキテクチャに適したアルゴリズムへ改良していく．

最新の GPU では，より汎用計算に最適化されたアーキテクチャに改良されている[5]．そのため従来の GPU では高速に動作させることが難しかった処理についても，GPU で高速に動作させることができないか検討する．

参考文献

1) 高道悦子，中野浩嗣: FPGA を用いた画像検索システム，信学技報, Vol.102, No.428, pp. 37-42 (2002).
2) NVIDIA: CUDA Programming Guide Version 2.3.1, (2009). http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf
3) 杉浦司: モルフォロジー演算を用いた類似画像検索の一考察，電子情報通信学会東海支部卒業研究発表会, O1-04 (2009).
4) Victor Podlozhnyuk: Histogram calculation in CUDA, NVIDIA GPU Computing SDK CUDA Advanced Topics Whitepaper, (2007). http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/histogram256/doc/histogram.pdf
5) NVIDIA: Whitepaper NVIDIA's Next Generation CUDA Compute Architecture: Fermi, (2010) http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf