PC 向けグラフィックカードを活用した画像処理の高速化

林 豊洋[1]

1 はじめに

可視画像に対する画像処理技術の研究開発は進歩が目覚ましく，多くの画像処理技術を応用した製品が登場しています．筆者も，動画像内に存在する物体を実時間追跡する技術や [1][2]，車載カメラが撮影した動画像から，先行車までの車間距離や自車の運動パラメータを推定する研究 [3][4] を推進しています．

これら画像処理を応用した技術の研究開発や実用化を行うためには,「単一の画像処理機能の高速，実時間性」が重要です．高速な画像処理を行うことができれば，研究開発では重要となる，多くのデータに対する十分な実験および検討を行うことが可能です．また，製品化等の実用化の段階においても，高速処理が可能な機能を組み合わせることにより，ビデオカメラ等で撮影した画像 (以下，入力画像と呼びます) を遅延無く処理し，即座に結果を返す応用製品が作成できます．

したがって，高速で実時間処理が可能な画像処理技術や，それをを提供するライブラリは，画像処理技術分野において非常に有用です．本研究では，入力画像を複数の小領域画像に分割し，各領域に対して並列に画像処理を行うことにより高速処理が可能なライブラリを提案します．並列化には，近年性能向上の目覚ましいハードウェアである GPU(Graphics Processing Unit) を活用しました．GPU は，本来は PC 等に接続するモニタ上で，グラフィックスを高速描画するために用いられる機器です．一見「画像処理の高速化」には関係ない機材に思えますが，GPU はグラフィックス処理を効率よく行うため，大規模な並列化が可能なハードウェア構成がなされています．このハードウェア構成は，画像処理の高速化に応用することが可能です．

また，PC で一般的に用いられる機器であるため，大規模な並列化が可能であるにも関わらず，安価に販売されています (本研究では，2 万円程度で市販されている GPU を用いています)．

本研究では GPU を活用し，画素レベルでの画像処理関数をライブラリとして実装しました．また，応用的な画像処理関数として，Harris Corner Detector[5] による画像のコーナー検出関数を実装しました．

これらの画像処理関数に対し，CPU のみで実装したプログラムとの処理時間を比較しました．比較実験の結果，入力画像の画素数が少ない場合のフィルタ処理 (畳み込み演算) およびヒストグラムに関する演算 (並列性が低い，排他制御を要する演算) においては CPU での実行が効率的でしたが，これら以外のほぼ全ての関数において，GPU による並列化が高速となる結果を得ることができ，画像処理の高速化が実現できました．

以下，2 章にて画像処理の高速化，並列化について述べ，3 章にて GPU を用いた並列化手法を紹介します．4 章にてライブラリとして実装した各種関数に関する詳細について述べ，5 章にて各種関数の実行速度等に関する比較実験を行います．6 章にて考察を行い，7 章にて本研究を総括します．

[1]情報科学センター　助教　toyohiro@isc.kyutech.ac.jp

2　並列化に基づく画像処理の高速化

現在，多くの画像処理技術を応用した製品が実用化されています．これら画像処理を応用した技術の研究開発や実用化には,「単一の画像処理機能の高速，実時間性」が必要となります．高速な画像処理を行うことができれば，研究開発では重要となる，多くのデータに対する十分な実験および検討を行うことが可能です．また，製品化等の実用化の段階においても，高速処理が可能な機能を組み合わせることにより，ビデオカメラ等で撮影した画像 (以下，入力画像と呼びます) を遅延無く処理し，即座に結果を返す応用製品が作成できます.

このような要求に対応するため，現在様々な画像処理ライブラリが実用化・製品化されています．これらのライブラリは計算アルゴリズムの工夫により，高速な画像処理関数を提供しているものの，基本的には単一の CPU で画像処理を逐次実行する仕組みが利用されています．したがって，多くの画素に対して処理を行う場合 (高解像度画像に対する処理) や，多くの処理すべき候補が存在する場合 (多くの候補点に対してマッチング，類似度演算等の処理を行う) には，実時間処理が困難となります.

画像処理の分野では，入力画像中の各画素に対して演算を行う画素レベルの処理手法や，多くの候補点に対して，同様の評価関数を適用し，評価値を求める手法が大半を占めています．画素レベルの処理には色空間の変換，フィルター処理 (エッジ処理) 等があり，評価値を求める手法には，候補点のパターンマッチングや固有値の演算処理等があります．これらの処理は，各画素や候補点に対して，すべて独立に演算を行うことができます.

従って，プログラム上の工夫を行えば，画像処理アルゴリズムは高い並列処理を行うことができ，逐次処理と比較して高速化が期待できます．本研究では，並列化に基づく画像処理の高速化手法に関して検討を行い，高速処理が可能なライブラリの構築を行います.

2.1　並列化の概要

本節では，画像処理の並列化に関して説明します.

画像処理の並列化には，データを並列に入力し処理する手法や，適用するアルゴリズムの内部を並列化し，データを逐次的に入力する手法など，様々な手法が適用可能です．前節で述べたとおり，多くの画像処理の手順は，各画素や候補点に対してすべて独立に演算を行うことが可能です．この特性を利用すると，下記の手順により並列化が実現できます (図 1)

1. 入力データを分割可能な複数の小領域に分割する
2. 各小領域画像を並列に処理する (それぞれの小領域内は逐次処理となる)
3. 各小領域での処理結果を統合する

2.2　並列化手法の特性

画像処理の並列化では，小領域の分割数を増やすほど、高い並列性を持たせることができます．各小領域は並列に処理が行えるよう，複数の演算装置に割り振られます．したがって，多くの演算装置を確保するための枠組み (並列化手法) が重要となり，様々な並列化手法・ハードウェア的な構成が実用化されています．本節では，並列化が可能な構成を例示し，それぞれの特性について紹介します.

CPU の SIMD 演算の活用　近年の PC に搭載されている演算装置 (CPU) には，複数のデータに対して同時に演算を実行する SIMD 演算機能 (Intel 社:SSE4，AMD 社:SSE5 等) が搭載されています [6]．しかし，本来 CPU は逐次的な演算を実行するために設計されているため，4 個の 32bit データに対して同時演算が行う程度の並列度にとどまるため，本研究には適さないと考えられます.

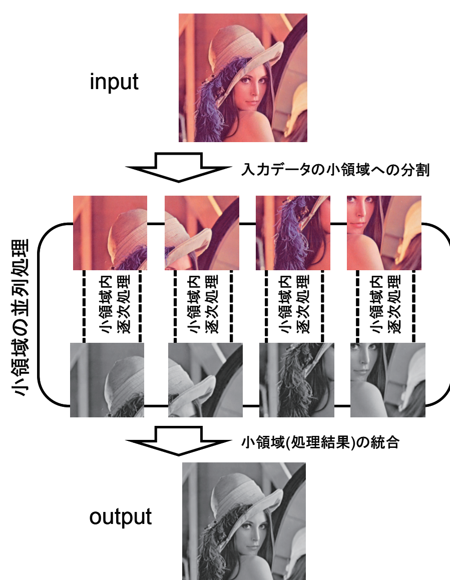

図 1: 並列化の概要

SMP(Symmetric Multi Processing) 構成の活用 近年は，単一のパッケージに複数の演算コアを有する安価な CPU が製品の大半を占めており (Intel 社:Core2，AMD 社:Phenom 等)，これらを活用すれば容易に SMP を構成できます．SMP による並列化の実装例として，プログラム中の繰り返し部分を OpenMP[7] を用いて分割し，任意の CPU に演算命令を割り当てる手法が考えられます．プログラムが利用するデータは同一計算機内の主記憶 (メインメモリ) に存在するため，データの分割や統合を高速に行うことが可能です．このように SMP 構成は容易に並列化が実現できますが，一般的に入手できる計算機では，演算コア数が 16 コア程度の構成 (= 並列度) が限度となります．

複数の計算機の活用 複数の計算機を用いてクラスタを構成し，各計算機にデータを分割することにより，処理の並列化を行うことが可能です．近年は，PC を用いた計算機クラスタと MPI[8] 等のデータ通信用のライブラリを利用することによる大規模な並列化が積極的に行われています．しかし，各計算機へのデータの送受信は外部との通信 (Ethernet や専用の結合網を利用) となるため，送受信による遅延時間を念頭に置く必要があります．どちらかといえば大規模化に適した手法であり，計算の高速化を実現するための設計は容易ではありません．

3 GPU を活用した画像処理の並列化

前節で例示した並列化手法は，演算装置で処理できる規模，割り当て可能な分割数，分割数を増やすことによる遅延時間の問題があり，画像処理に対してバランスの良い手法とは言い難いものとなります．本研究では，バランスの良いハードウェアとして，近年性能向上の目覚ましいハードウェアである GPU(Graphics Processing Unit) に着目し，画像処理を高速化する手法を提案します．

GPU とは，3 次元グラフィックの描画を支援する機能などを有するハードウェアであり，PC に接続するモニタ上でグラフィックスを高速描画するために用いられています．パーソナルコンピュータ向けの GPU は NVIDIA 社 (GeForce シリーズ) や AMD(ATI) 社 (Radeon シリーズ) によって開発されています．グラフィックス描画向けの機器であるため，一見「画像処理の高速化」には関係ないように思えますが，GPU は高速な浮動小数点演算回路や，大規模な並列化が可能な機器構成がなされており，これ

らの機能は画像処理に応用することが可能です．また，PC で一般的に用いられる機器であるため，大規模な並列化が可能であるにも関わらず，安価に導入できます．表 1 に，本研究で用いた GPU の仕様を示します．

表 1: GPU の性能例 (Nvidia 社 GeForce 8800GT)

クロック周波数	1.8GHz
演算コア数	14
並列実行命令数	7,168
処理性能	336GFlops
主記憶 (VRAM) 容量	512MByte
主記憶帯域幅	57.6GByte/sec
PC とのバス	PCI-express 2.0
PC とのバス帯域幅	8GByte/sec

このような GPU の並列処理に関する性能が近年注目されており，GPU を用いた汎用的な並列プログラミング手法は GPGPU[9] と呼ばれています[2]．

なお，並列計算を行うための GPU で実行されるプログラムの作成には，従来 (2006 年頃まで) は 3 次元グラフィックス用のライブラリ (OpenGL，Direct3D) を利用し，ピクセルシェーダと呼ばれるグラフィックスレンダリングの手法を用いる必要がありました．現在は NVidia 社より，C 言語を拡張した文法とライブラリによる GPU プログラムの実行および作成環境 (CUDA : Compute Unified Device Architecture) が提供されており [10]，容易に GPU を利用することが可能となりました．

GPU による画像処理の流れ

GPU による画像処理は，以下の手順で実行されます (図 2)．

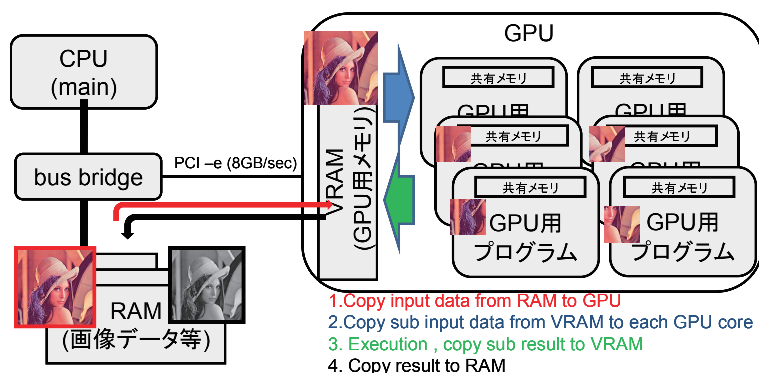

図 2: GPU による処理の並列化

1. 入力データを計算機内の主記憶から GPU 内の主記憶へ転送
2. GPU で実行される画像処理プログラムを GPU へ転送 (CUDA では自動的に実行される)

[2]様々なアルゴリズムを実装する研究が始まっています [11][12]

3. GPU で画像処理プログラムが実行され，結果が GPU 内の主記憶へ保存される

4. GPU 内の主記憶から計算機内の主記憶へ処理結果を転送

上記の手順の通り，GPU で画像処理を行う場合，計算機と GPU 内の主記憶間において，入力データおよび処理結果の転送が必要となります．したがって，PC 本体とのバス帯域幅が処理性能に大きく影響します．表 1 に示す GPU の性能によると，バス帯域幅は 8.0GByte/sec と非常に高速です．近年の PC に搭載される CPU と主記憶間の帯域幅は 10.6GByte/sec 程度であり，ほぼ近い性能を示していることから，データの転送に関する影響はわずかと言えます．

また，GPU の動作クロック周波数は近年の CPU のクロック周波数と比較して低速ですが，演算コアは 14 基搭載されており，同時に実行可能な命令数は 7,000 程度に達しており，極めて高い並列演算性能を有する設計となっています．このような並列性の高い設計がなされた GPU の演算性能は約 300GFlops であり，これは一般的に入手できる高速な PC 向け CPU の演算性能の 7 倍程度となります[3].

これらの理由より，GPU によって高い並列性を有するプログラムを記述することができれば，従来 CPU 単体で行っていた画像処理の高速化が期待できます．

4　GPU を用いた画像処理ライブラリの実装

本研究では GPU を活用した画素レベルでの画像処理関数を実装し，外部プログラムから利用可能な関数ライブラリを構築します．

ライブラリは並列化の対象によって以下の 3 種類に分類し，構築を行います．分類とそれぞれの対象において実装を行った機能は以下の通りです．

1. 入出力が 1 対 1 の並列化 入力画像の各画素およびデータ配列内の各要素を並列化の対象として処理を行います．機能として,「色空間の変換」「エッジ抽出」「フィルタ処理」「2×2 対称行列の固有値算出」を実装します．これらは，入力と出力を 1 対 1 の関係で処理できるため，単純なアルゴリズムで実装が可能です (図 3(a)).

2. 処理結果の統合を要する並列化 データ配列を入力し，配列内の各要素を並列化の対象として処理を行います．上記の対象 1. における並列化と異なり，最終的に処理結果を統合し出力データを生成する必要があるため，排他制御等を行う必要が生じます (図 3(b))．機能として,「ヒストグラムの作成」「ヒストグラム間の距離演算」「2 つの入力データの正規化相関演算」を実装します．

3. 応用処理に対する並列化 上記の並列化手法を組み合わせ，より応用的な画像処理関数を構築します．機能として，入力画像内に存在するコーナーを検出する手法である「Harris Corner Detector」を実装します．

なお，開発環境には CUDA 1.1[4]を利用し，Microsoft Windows 上で動作するライブラリを構築します (詳細は 4.5 節にて紹介します)．また，GPU には CUDA による実装が可能である NVidia GeForce 8800GT を利用します．

4.1　入出力が 1 対 1 の並列化処理

4.1.1　色空間の変換，エッジ抽出

色空間の変換とエッジ抽出は，ある入力画像内の画素 $I_{in}(x, y, c)$ から処理結果 $I_{out}(x, y, c)$ を並列に計算することができます．

[3]2010 年 2 月時点では，GPU の演算性能は 1TFlops に達しており，より高速化しています．

[4]2010 年 2 月時点では，倍精度演算のサポートや主記憶管理方法の利便性が向上した，CUDA 2.3 が利用できます．

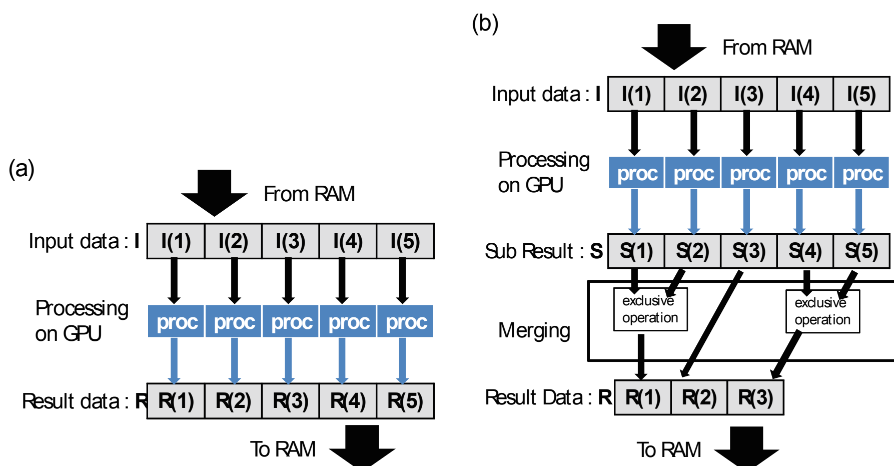

図 3: 各対象毎の並列化手法

色空間の変換は，RGB 色空間から輝度成分，YUV，HSI，HSV，rgb の各色空間への変換関数を実装します．GPU では，入力された RGB 色空間で記述された画素に対して，各色空間への変換式によって変換を行います．

またエッジ抽出は，入力された輝度成分で記述された画素に対して，Sobel，Prewitt，LoG の各種フィルタを適用します．なお，Sobel，Prewitt の一次微分フィルタに関しては，縦横のエッジ成分とエッジの方向成分を算出できる関数を実装します．

4.1.2 フィルタ処理

フィルタ処理は，前述のエッジ抽出フィルタを一般化した処理であり，任意のサイズで記述されたフィルタを画像に適用する処理となります．これは，任意のフィルタの畳み込み処理となるため，入力画像 I とフィルタ C をフーリエ変換し，フーリエ領域での積を算出することによってフィルタ処理を行います．

このようなフィルタ処理は下記の式で定義できます．

$$\begin{aligned} Filter(I,C) &= I \otimes C \\ &= IFFT(FFT(I) \times FFT(C)) \end{aligned}$$

なお，フーリエ変換には CUDA 内で提供されている FFT ライブラリ (CUDAFFT) を利用します．

4.1.3 2×2 対称行列の固有値算出

サイズが 2×2 の対称行列の固有値は，後述の Harris Corner Detector にて，各画素がコーナー領域であるかを判定する場合等に利用されます．

本研究では，入力された一つの 2×2 対称行列に関して，第一固有値及び第二固有値と，対応する固有ベクトルを算出します．これら固有値固有ベクトルの算出には，数値計算法であるヤコビ法を利用します．

4.2　処理結果の統合を要する並列化処理

4.2.1　ヒストグラムの作成

ヒストグラムとは，入力データの持つ値に関する度数分布のことを指します．入力データが画像の場合，画像の持つ輝度や色分布のヒストグラムが利用されます．

本研究では，画像の輝度ヒストグラムの作成を想定した処理を実装します．ヒストグラムの作成は，各画素の持つ輝度に応じたヒストグラム保存用の領域に投票することによって行います．

ここで，複数の画素に対して並列にヒストグラム作成処理を行った場合，共有のヒストグラム保存領域に同時書き込みが発生します．同時書き込みが発生した場合は正しい投票結果が得られないため，投票時に排他制御が必要となります (図 4)．

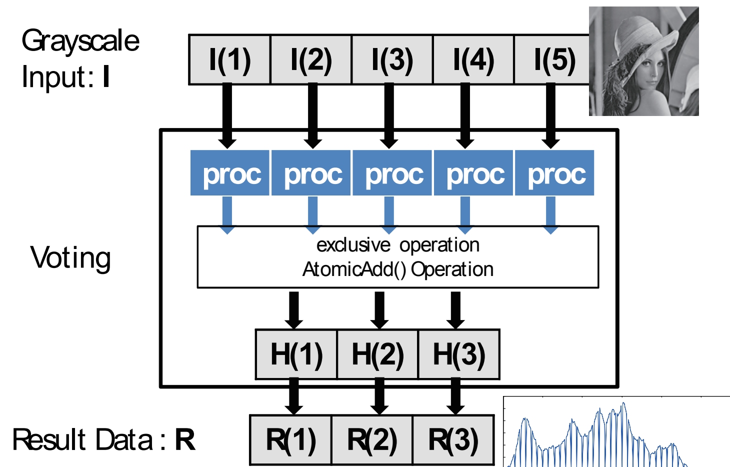

図 4: ヒストグラムの作成

本研究では，CUDA が備えるアトミック加算命令 (排他的に指定したデータ領域の値を加算する命令) である `AtomicAdd` 命令により排他制御を行い，ヒストグラムを作成します．

4.2.2　ヒストグラム間の距離演算，正規化相関演算

二つのヒストグラム間の距離演算や入力データの相関演算は，テンプレートマッチングにおいて類似度演算の手段として利用されます．

本研究では，ヒストグラム間の距離演算としてヒストグラムインタセクションを実装し，入力データの相関演算として正規化相関を実装します．ヒストグラム $H1$ と $H2$ 間のインタセクション $HIN(H1, H2)$ および入力データ $D1$ と $D2$ 間の正規化相関 $CORR(D1, D2)$ は，それぞれ以下の式で定義できます．

$$HIN(H1, H2) = \sum_{i=1}^{i\max} \min(H1(i), H2(i))$$

$$CORR(D1, D2) = \frac{\sum_i (D1_i - \bar{D}1)(D2_i - \bar{D}2)}{\sqrt{\sum_i (D1_i - \bar{D}1)^2 \sum_i (D2_i - \bar{D}2)^2}}$$

これらの演算を並列化する場合，ヒストグラムインタセクションはヒストグラムの各要素毎の最小値演算が並列化の対象となり，正規化相関では内積を算出する際の要素毎の積演算が並列化の対象となります．

4.3 応用処理に対する並列化

4.3.1 Harris Corner Detector

並列処理を活用した応用処理として，本研究ではコーナー検出手法である Harris Corder Detector を実装します．Harris Corner Detector は各画素において，近傍領域に対する縦，横，斜めの勾配を観測し，各画素がコーナーに相当するかを推定できる手法です (図 5).

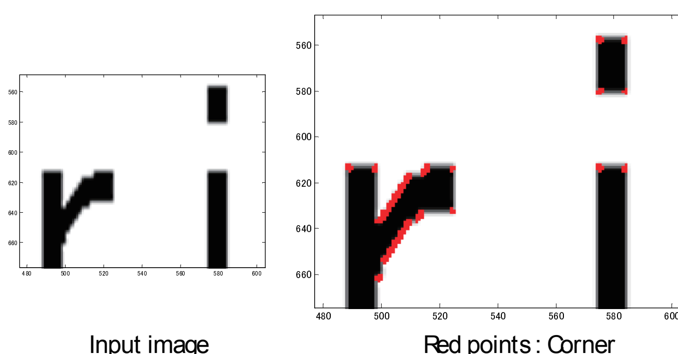

図 5: Harris Corner Detector

処理の手順は，

1. カラー画像 C の輝度画像 I への変換
2. 各画素の近傍領域に対する縦 $(I_{xx} = (\frac{\partial I}{\partial x})^2)$，横 $(I_{yy} = (\frac{\partial I}{\partial y})^2)$，斜め $(I_{xy} = \frac{\partial I}{\partial x}\frac{\partial I}{\partial y})$ 勾配の算出
3. 各勾配画像に対する Gaussian フィルタの適用 $(A = G \otimes I_{xx}), (B = G \otimes I_{yy}), (C = G \otimes I_{xy}$ の算出$)$
4. 各画素の勾配よりヘッセ行列 $H_i = \begin{pmatrix} A_i & C_i \\ Ci & Bi \end{pmatrix}$ を作成，固有値 λ_1, λ_2 の算出
5. 固有値を用いた評価値 $M_i = \lambda_1\lambda_2 - \alpha \times (\lambda_1 + \lambda_2)^2$ の算出，コーナーの判定

となります．これらの手順は，全てが本節の冒頭で分類した「入出力が 1 対 1 の並列化処理」に相当するため，それぞれの手順を分離し，関数として GPU 上に実装することが可能です．

4.4 複数の GPU を用いた並列性の向上

上記で述べた各種関数は，最大で 7168 個のデータ列が並列に演算できます[5]．この数を超えたデータに関しては，並列処理を繰り返すことによって演算が行われます．例として，640×480 画素の画像データを入力した場合において，7168 個のデータを並列処理できた場合は，計算を完了するために 43 回の繰り返し処理が必要です．繰り返し処理は処理時間の増大を招くため，さらなる高速化を行うためにはこれを減らすことが必要となります．すなわち，回数を減らすためには，並列に処理するデータ数を増やす必要が生じます．

[5] GPU が持つ最大の並列実行命令数に依存します

本研究では，並列するデータ数を増やす方法として，複数の GPU を用いる手法を検討します．具体的には，計算機に複数個の GPU を取付け，各 GPU に割り当てるデータ列を CPU プログラムにて分割し，GPU1 基あたりに転送するデータ量を削減します (図 6)．

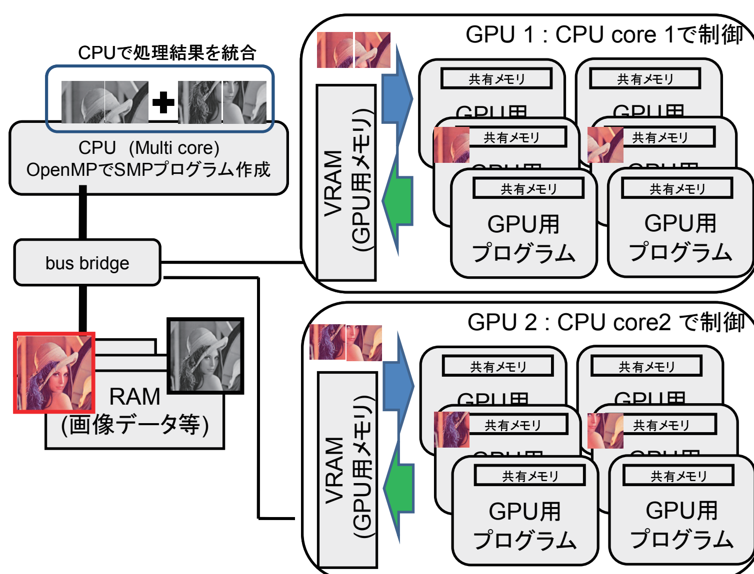

図 6: 複数の GPU を用いた並列性の向上

2 基の GPU を利用した場合，上記の例と同じ画像データに対して，14336 個のデータが並列処理できるため，22 回の繰り返し処理で計算が完了します．

4.5　ユーザプログラムからの呼び出し

提案するライブラリの実装は，データの入出力や GPU の制御部等の CPU で実行するコードは C 言語を利用し，画像処理の並列実行を行うための GPU コードは CUDA を利用します．また，ユーザプログラムからの呼び出しが容易に行えるよう，Microsoft Windows 上で利用可能な DLL(Dynamic Link Library) 形式で実装を行います．

DLL 形式で作成されたライブラリは，C 言語からの利用以外に，画像処理プログラムにおいて利用される Matlab 環境からも容易に呼び出し可能です[6]．

5　評価

本研究にて構築した GPU を用いた画像処理ライブラリの処理性能を評価します．評価は，テスト用画像およびデータ列を入力データとし，要した処理時間を計測することによって行います．

入力データとして，4 種類の異なる解像度 $(256 \times 256, 512 \times 512, 1024 \times 1024, 2048 \times 2048)$ の画像データを用います．各処理関数において，色空間の変換は RGB 色空間から HSV 色空間への変換，エッジ抽出は各方向成分の Sobel オペレータおよびエッジの方向成分の算出，フィルタ処理は 7×7 のウィンドウサイズにおける Gaussian フィルタ処理，ヒストグラムの作成は長さ 256 階調の輝度ヒストグラムの作成，ヒストグラムの距離演算はヒストグラムインタセクションの算出を実行します．

[6]Matlab には，DLL を呼び出すための命令 (loadlibrary 命令) が用意されています

また，2×2 対称行列の固有値を求める関数の入力データには，ランダムに生成した数値で構成された，4 種類の個数 $(256 \times 256, 5120 \times 512, 1024 \times 1024, 2048 \times 2048)$ の行列を用います．

処理時間の計測は，3 つの環境 (Matlab による実装 (CPU による処理)，GPU による実装，GPU2 基による実装 (CUDA が提供する線形代数ライブラリを用いるフィルタ処理を除く))) において行います．GPU での処理関数は，Matlab から DLL 経由で呼び出すことによって実行します．なお，GPU2 基による実装において，ヒストグラム処理と正規化相関演算は，CPU を用いて各 GPU における演算結果を統合しているため，純粋な GPU のみを用いた実装ではありません．したがって観測された処理時間は参考結果とします (表 2~5 に括弧書きで処理時間を記述します)．

実験環境の主な仕様は，Matlab が動作する計算機の CPU は Intel Core2 Quad Q6600 (処理性能約 40GFlops)，主記憶は 4GByte，CPU と主記憶とのバス帯域幅は 10.6GByte/sec，計算機と GPU とのバス帯域幅は 8GByte/sec(PCI-Express) となります．

また，全ての画像処理関数において，CPU を用いて実装した処理関数との計算誤差は無視できる量であることを確認しています．

表 2~5 に，各解像度および各環境における処理時間の計測結果を示します．

表 2: 処理時間 (データ長：256×256)，単位：msec

処理関数	CPU	GPU	GPU×2
色空間の変換	44.31	9.620	**6.941**
エッジ抽出	42.49	16.44	**13.92**
フィルタ処理	**1.988**	6.922	–
ヒストグラム作成	**7.092**	17.19	(9.302)
ヒストグラム距離	**0.012**	0.978	(0.651)
正規化相関	8.070	**3.482**	(3.140)
固有値算出	158.0	17.47	**11.21**
コーナー検出	551.3	61.48	**53.10**

表 3: 処理時間 (データ長：512×512)，単位：msec

処理関数	CPU	GPU	GPU×2
色空間の変換	187.6	30.27	**19.79**
エッジ抽出	198.1	63.98	**51.60**
フィルタ処理	17.31	**13.73**	–
ヒストグラム作成	**26.34**	68.74	(45.70)
ヒストグラム距離	**0.013**	1.292	(0.716)
正規化相関	18.09	**8.660**	(8.547)
固有値算出	625.8	55.45	**34.93**
コーナー検出	2295	163.2	**130.9**

表 4: 処理時間 (データ長：1024 × 1024)，単位：msec

処理関数	CPU	GPU	GPU×2
色空間の変換	774.7	132.9	**79.65**
エッジ抽出	862.1	259.0	**213.6**
フィルタ処理	89.84	**47.85**	–
ヒストグラム作成	**103.4**	274.8	(168.8)
ヒストグラム距離	**0.014**	1.350	(0.7697)
正規化相関	60.83	**30.29**	(28.65)
固有値算出	2529	199.0	**124.3**
コーナー検出	9062	547.0	**447.5**

表 5: 処理時間 (データ長：2048 × 2048)，単位：msec

処理関数	CPU	GPU	GPU×2
色空間の変換	3033	398.3	**336.4**
エッジ抽出	3598	1027	**851.9**
フィルタ処理	399.0	**224.0**	–
ヒストグラム作成	**429.5**	1103	(568.6)
ヒストグラム距離	**0.016**	1.364	(0.7170)
正規化相関	232.1	**117.0**	(115.4)
固有値算出	10163	793.6	**484.3**
コーナー検出	36405	2284	**1853**

6　考察

処理時間を計測した結果，入力データが 256 × 256 でのフィルタ処理とヒストグラムの作成，インタセクションの演算処理を除き，GPU による処理が高速である結果を得ました．GPU による処理速度が CPU と比較して低下する原因は，以下の 2 点が考えられます．

1. GPU で処理する手順が比較的単純で入力データが少ない場合は，画像処理に要する処理時間に対し，入力データと処理結果を CPU と GPU の主記憶間で転送するオーバーヘッドが無視できなくなる
2. 評価値の和を求める演算や処理結果の統合に用いる排他制御は逐次処理であるため，GPU の持つ並列性が働かない

上記の処理以外は，GPU による画像処理が高い処理性能を示しました．CPU との処理時間の差が最も大きい処理はコーナー検出処理で，CPU による処理に対して約 16 倍の高速処理を実現しました．

また，GPU1 基による処理と GPU2 基による処理性能を比較すると，最大 1.67 倍の性能向上を示し，複数の GPU による計算が有効な処理が存在することがわかりました．ただし，複数回のデータ転送が必要となる命令 (データ長が長い，複数の関数を実行する命令等) に関しては，性能向上が 1.15 倍程度にとどまる結果となりました．原因として，2 基の GPU はデータの送受信時に計算機とのバスを共有するため，複数回のデータ転送を行う際に，バスの競合が生じてしまったことが挙げられます．

6.1　GPUによる並列化の制約・課題

実験によりGPUによる並列化は高速な画像処理が実現できることが示されましたが，CPU向けのプログラムと比較して，GPU向けのプログラムには設計上の制約が大きいことがわかりました．

GPU向けのプログラムは，GPU本体に全てのプログラムが転送され実行されます．CPU向けのプログラムのように，プログラムを主記憶に置き逐次実行することはできないため，GPUの回路規模を上回るような大規模な処理関数は実装できません．同様の理由により，再帰呼び出し等も実行できません．従って，複雑な処理をGPUで行う場合は，一部の処理をCPU側で行い，GPUを併用するといったプログラムの設計を考慮する必要があります．また，複数のGPUを用いる場合は，バスの競合を避けるための効率的なデータの転送が必要であり，転送方法の改良が課題となります．

7　まとめ

本研究では画像処理の高速化を行うため，グラフィックス描画用のハードウェアであるGPUに注目し，GPUにおける画像処理の並列化に関して検討を行いました．また，DLL形式でのライブラリとして実装を行いました．提案したライブラリでは，色空間の変換等の基本的な画像処理関数やパターンマッチングで利用される正規化相関，応用処理としてHarris Corner Detectorを用いたコーナー検出関数を実際に実装し，その動作を確認しました．処理時間の計測を行った結果，排他制御を用いず，入力データ長が 512×512 を超える処理においては，GPUでの処理が有効であることがわかりました．

本研究で利用したGPUはPCで利用される一般的な機器であるため，容易かつ安価に入手できます．このような一般的な機器において大規模な並列化が行えることから，ご自身の研究へ活用してみてはいかがでしょうか．

参考文献

[1] ”尤度分布の形状を用いた物体追跡の安定化”，林 豊洋，榎田 修一，江島 俊朗画像電子学会誌第35巻第5号，pp.582-587，2006.

[2] ”尤度分布の観測と分割に基づく物体追跡”，林 豊洋，榎田 修一，江島 俊朗，第6回情報科学技術フォーラム，第H巻，pp.17-20，2007.

[3] ”車間距離推定のためのドライブレコーダ画像における先行車追跡”，榎田 修一，林 豊洋，江島 俊朗，画像ラボ 2007年10月号，pp.53-57，2007.

[4] ” ドライブビデオレコーダ画像解析に基づく運転軌跡の復元”，岡野 謙二, 林 豊洋, 榎田 修一, 江島 俊朗，第10回画像の認識・理解シンポジウム，pp.1283-1288，2007.

[5] ”A combined corner and edge detector”，C. Harris and M. Stephens,，Proceedings of the 4th Alvey Vision Conference，pp.147-151，1988.

[6] ”Intel Streaming SIMD Extensions 4 (SSE4) Instruction Set ”，Intel Corp., http://www.intel.com/technology/ architecture-silicon/sse4-instructions/，2007.

[7] ”The OpenMP specification for parallel programming”，OpenMP Architecture Review Board, http://www.openmp.org/.

[8] ”Message Passing Interface Forum”，MPI Forum，http://www.mpi-forum.org/.

[9] ”General-Purpose Computation Using Graphics Hardware”，http://www.gpgpu.org/.

[10] ”NVIDIA CUDA Zone”，NVIDIA Corp.，http://www.nvidia.com/object/cuda_home.html, 2007.

[11] ”GPU-based implementation of the KLT Tracker”, http://cs.unc.edu/~ssinha/Research/GPU_KLT/.

[12] ”GPU-based implementation of Scale Invariant Feature Transform”, http://cs.unc.edu/~ccwu/siftgpu/.