HI2009-38
ME2009-38
AIT2009-38

GPUを用いた位相限定相関法の高速化

松尾堅太郎† 三好 正之† 濱田 剛† 柴田裕一郎† 正田 備也†
小栗 清†

† 長崎大学工学部 〒852–8521 長崎県長崎市文教町 1–14
E-mail: †{aniki,miyoshi,shibata,oguri}@pca.cis.nagasaki-u.ac.jp, ††{hamada,masada}@cis.nagasaki-u.ac.jp

あらまし 位相限定相関法は画像マッチング・画像レジストレーションにおいて高いロバスト性とサブピクセル単位での高い精度を実現する計算方法であるが同時に計算コストが膨大であるという側面もある．これまで位相限定相関法の高速化には専用 LSI や FPGA を用いた方法が試みられてきた．今回我々は新たに GPU(Graphics Processing Unit) を用いた位相限定相関法の高速化手法を考案し，Nvidia GPU, GeForce8800GTS へ実装を行った．GPU 1 台当たりの処理時間に 256×256 pixel 画像が 2.36 秒, 512×512 pixel 画像が 7.92 秒，1024×1024 pixel 画像が 27.65 秒で処理可能なことを確認し，これが過去の専用 LSI や FPGA を用いた場合の計算速度と比較して約 10 倍程度高速であることを確認した．

キーワード 位相限定相関法， GPU， CUDA， 画像レジストレーション

Accelerating the Phase Only Correlation method using GPUs

Kentaro MATSUO†, Masayuki MIYOSHI†, Tsuyoshi HAMADA†, Yuichiro SHIBATA†, Tomonari MASADA†, and Kiyoshi OGURI†

† Faculty of Engineering, Nagasaki University Bunkyo-machi 1–14, Nagasaki-shi, Nagasaki, 852-8521 Japan
E-mail: †{aniki,miyoshi,shibata,oguri}@pca.cis.nagasaki-u.ac.jp, ††{hamada,masada}@cis.nagasaki-u.ac.jp

Abstract The Phase Only Correlation (POC) method demonstrates high robustness and subpixel accuracy in the pattern matching and the image registration. However, there is a disadvantage in computational speed because of the calculation of 2D-FFT etc. We have proposed a novel approach to accelerate POC method using GPU to solve the calculation cost problem. Using our GPU-based POC implementation, each POC calculation can be done within 2.36 seconds for 256 × 256 pixels, within 7.92 seconds for 512×512 pixels, and 27.65 seconds for 1024×1024 pixels.

Key words Phase Only Correlation, GPU, CUDA, registration

1. はじめに

画像をあらかじめ用意された参照パターンと照合し，類似度や位置ずれなどの情報を出力する処理を画像マッチングと言う．現在，画像マッチング技術は生体認証における指紋照合装置や，産業用機械における位置決め用センサなどに実用されており，コンピュータビジョンの分野で基本的な処理となっている．

画像マッチングの手法として，パターン間の残差を用いる方法，各種相関関数を用いる方法，フーリエ変換を利用する方法など，これまでに様々な手法が提案されている [1]．近年，その一つである位相限定相関法 (Phase Only Correlation: POC) が注目を集め始めている．

位相限定相関法は，入力信号の位相情報のみに限定し相関を計算する手法であり，高いロバスト性とサブピクセル単位での高い位置合わせ精度を発揮する．位相情報を用いる画像マッチングの研究の起源としては，Kuglin らによる研究が 1975 年に発表されている [2]．この研究は後に，回転・拡大縮小の計測などに応用されている [3]．ただし，入力信号から位相情報を得る際に二次元のフーリエ変換を必要とするため，当時は計算コストの面で実用化は難しかったが最近になって半導体技術の進歩にともない指紋照合装置や，3 次元計測等への応用が盛んに行われている [4]，

ハードウェアへの実装による位相限定相関法の高速化としては，256 × 256 画素の POC 処理を 27.15ms で処理可能である専用 LSI が開発されている [5]．また，位相限定相関法を用いた指紋認証アルゴリズムの FPGA 実装 [6] や，位相限定相関法

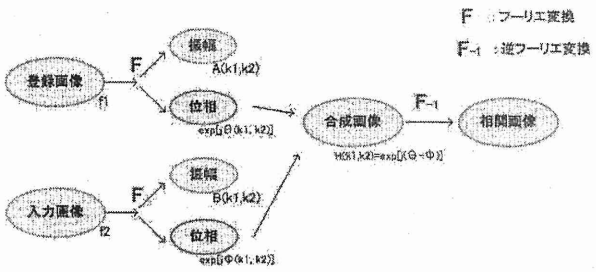

図 1　位相限定相関法のアルゴリズム

に基づいたビジョンアルゴリズムの FPGA 実装 [7] なども報告されている．

これに対し，筆者らは位相限定相関法の GPU(Graphics Processing Unit) による高速化を提案した．GPU は，数値データから CG を描画するレンダリング処理に特化したハードウェアである．

近年の GPU の処理性能の向上にともない，GPU をレンダリング処理以外の汎用計算にも応用することが可能になってきた．GPU を汎用計算に利用するための方法は複数存在するが，最も広く用いられている方法として，Nvidia 社の CUDA(Compute Unified Device Architecture) がある．これは 2006 年 11 月に Nvidia 社が発表した Nvidia 製 GPU のための C 言語統合開発環境である．本稿では CUDA を用いて位相限定相関法による画像の平行・回転移動量の推定計算を GPU に実装した．

2. 位相限定相関法のアルゴリズム

本稿では，平行ずれに対応する位相限定相関法 (POC)，回転ずれに対応する回転不変位相限定相関法 (RotationInvariant Phase Only Correlation: RIPOC) の 2 種類の画像処理アルゴリズムを GPU 上に実装した．以下にこれらのアルゴリズムの概要を示す．

2.1 POC

画像の位相成分には，形態に関する情報が多く含まれている．POC はこの性質を利用した手法である．POC では，入力信号に対しフーリエ変換をほどこした後，振幅成分を固定値に置き換え相関をとることで，識別性能を向上させている．

POC のアルゴリズムを図 1 に示す．図 1 から分かるように，POC アルゴリズムは 3 つの二次元 FFT から成る，明解な構造を持つアルゴリズムである．

以下に POC の処理フローを示す．

[ステップ 0 (定義)]

$\tilde{f}(x_1,x_2)$ で表される連続空間で定義された 2 次元画像を考える．x_1,x_2 は実数である．δ_1，δ_2 を x_1，x_2 方向への微小移動量を表す実数とすると，$\tilde{f}(x_1,x_2)$ を x_1，x_2 方向へそれぞれ微小シフトした画像は $\tilde{f}(x_1-\delta_1,x_2-\delta_2)$ と表すことができる．これら 2 つの画像を標本化間隔 T_1，T_2 で標本化した離散空間における 2 次元画像をそれぞれ $f(n_1,n_2)$，$g(n_1,n_2)$ とし，次式のように定義する．

$$f(n_1,n_2) = \tilde{f}(x_1,x_2)\,|_{x_1=n_1T_1,x_2=n_2T_2} \tag{1}$$

$$g(n_1,n_2) = \tilde{f}(x_1-\delta_1,x_2-\delta_1)\,|_{x_1=n_1T_1,x_2=n_2T_2} \tag{2}$$

ただし，説明の簡単化のため，$n_1 = -M_1,\cdots,M_1$，$n_2 = -M_2,\cdots,M_2$ とし，画像サイズを $N_1 = 2M_1+1$，$N_2 = 2M_2+1$ とする．

[ステップ 1]

離散空間画像 $f(n_1,n_2)$ および $g(n_1,n_2)$ の二次元フーリエ変換 (DFT=Discrete Fourier Transform)$F(k_1,k_2)$，$G(k_1,k_2)$ を求める．二次元 DFT は次式で表される．

$$\begin{aligned} F(k_1,k_2) &= \sum_{n_1,n_2} f(n_1,n_2)W_{N_1}^{k_1n_1}W_{N_2}^{k_2n_2} \\ &= A_F(k_1,k_2)e^{j\theta_F(k_1,k_2)} \end{aligned} \tag{3}$$

$$\begin{aligned} G(k_1,k_2) &= \sum_{n_1,n_2} g(n_1,n_2)W_{N_1}^{k_1n_1}W_{N_2}^{k_2n_2} \\ &= A_G(k_1,k_2)e^{j\theta_G(k_1,k_2)} \end{aligned} \tag{4}$$

ここで，$k_1 = -M_1,\cdots,M_1$，$k_2 = -M_2,\cdots,M_2$ は離散周波数インデックスであり，回転因子を $W_{N_1}^{k_1n_1} = e^{-j\frac{2\pi}{N_1}k_1n_1}$，$W_{N_2}^{k_2n_2} = e^{-j\frac{2\pi}{N_2}k_2n_2}$ とする．また，$\sum_{n_1,n_2} = \sum_{n_1=-M1}^{M_1}\sum_{n_2=-M2}^{M_2}$ である．$A_F(k_1,k_2)$，$A_G(k_1,k_2)$ は振幅スペクトルであり，$e^{j\theta_F(k_1,k_2)}$，$e^{j\theta_G(k_1,k_2)}$ は位相スペクトルである．

[ステップ 2]

$F(k_1,k_2)$ と $G(k_1,k_2)$ の正規化相互パワースペクトル $R(k_1,k_2)$ を次式により計算する．

$$\begin{aligned} R(k_1,k_2) &= \frac{F(k_1,k_2)\overline{G(k_1,k_2)}}{|F(k_1,k_2)\overline{G(k_1,k_2)}|} \\ &= e^{j\{\theta_F(k_1,k_2)-\theta_G(k_1,k_2)\}} \end{aligned} \tag{5}$$

ここで $\overline{G(k_1,k_2)}$ は $G(k_1,k_2)$ の複素共役を表す．$e^{j\{\theta_F(k_1,k_2)-\theta_G(k_1,k_2)\}}$ は 2 つの画像の位相スペクトルの差である．

[ステップ 3]

位相限定相関関数 $r(n_1,n_2)$ を次式により求める．

$$r = \frac{1}{N_1N_2}\sum_{k_1,k_2} R(k_1,k_2)W_{N_1}^{-k_1n_1}W_{N_2}^{-k_2n_2} \tag{6}$$

$r(n_1,n_2)$ は正規化相互パワースペクトルの二次元逆離散フーリエ変換 (IDFT=Inverse DFT) である．また，$\sum_{k_1,k_2} = \sum_{k_1=-M1}^{M_1}\sum_{k_2=-M2}^{M_2}$ である．

[ステップ 4]

サブピクセル単位の移動量を求めるため，$r(n_1,n_2)$ に対し，相関ピークモデル式

$$\dot{r}(n_1,n_2) \simeq \frac{\alpha}{N_1N_2}\frac{\sin\{\pi(n_1+\delta_1)\}}{\sin\{\frac{\pi}{N_1}(n_1+\delta_1)\}}\frac{\sin\{\pi(n_2+\delta_2)\}}{\sin\{\frac{\pi}{N_2}(n_2+\delta_2)\}} \tag{7}$$

のフィッティングを行う．微小移動量 δ_1，δ_2 を少数単位で変化させ，その都度フィッティングを行い，もっとも RMS 誤差が小さくなる δ_1，δ_2 の値を求める．この時の δ_1，δ_2 がサブピクセル単位での移動量である．

また，上式の α は相関ピークの高さを表すパラメータであり，画像にノイズが加わると $\alpha<1$ の値をとる．

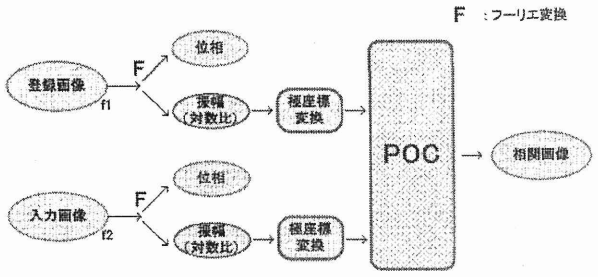

図 2　回転不変位相限定相関法のアルゴリズム

[ステップ 3] で得られた相関ピーク $r(n_1, n_2)$ の高さは二つの画像の類似度を表しており，二つ画像が同一である場合に相関ピークの高さは 1 となる．一方，相関ピークの座標は二つの画像の相対的な位置ずれに相当する．

2.2 RIPOC

POC では，画像の位相成分に注目して平行ずれを求める．これは画像の位相成分には平行ずれ情報が含まれているという性質を利用したものであるが，逆に考えれば，画像の振幅成分には回転・拡大情報が多く含まれることになる．RIPOC では振幅が持つこの性質を利用する．

RIPOC のアルゴリズムを図 2 に示す．RIPOC は，極座標変換により回転移動を平行移動へと帰着させた POC である．具体的には，フーリエ変換により得られる振幅スペクトルを対数変換し，極座標変換するという前処理を行う．次に，極座標変換されたデータを画像とみなし，POC 処理を行う．対数変換はダイナミックレンジを圧縮するために必要な処理である．一般に，自然画像の振幅スペクトルは低周波領域にエネルギーが集中するため，対数変換を行うことで全周波数領域の情報を均等に利用することができる．

以下に RIPOC の処理フローを示す．

[ステップ 0 (定義)]

2.1 の [ステップ 0] と同様に $\tilde{f}(x_1, x_2)$ で表される連続空間で定義された 2 次元画像を考える．$\tilde{f}(x_1, x_2)$ を角度 θ だけ回転し，x_1，x_2 方向にそれぞれ δ_1，δ_2 だけ微小移動した画像を $\tilde{g}(x_1, x_2)$ とする．これら 2 つの画像を標本化間隔 T_1，T_2 で標本化した離散空間における 2 次元画像をそれぞれ $f(n_1, n_2)$，$g(n_1, n_2)$ とし，次式のように定義する．

$$\begin{aligned} f(n_1, n_2) &= \tilde{f}(x_1, x_2)\,|_{x_1=n_1T_1, x_2=n_2T_2} \\ g(n_1, n_2) &= \tilde{g}(x_1, x_2)\,|_{x_1=n_1T_1, x_2=n_2T_2} \\ &= \tilde{f}(\dot{x_1}-\delta_1, \dot{x_2}-\delta_2)\,|_{x_1=n_1T_1, x_2=n_2T_2} \end{aligned} \tag{8}$$

ここで，$\dot{x_1} = x_1\cos\theta - x_2\sin\theta$，$\dot{x_2} = x_1\sin\theta - x_2\cos\theta$ である．

以下では，説明の簡単化のため $N = N_1 = N_2$，$M = M_1 = M_2$，$N = 2M + 1$ とする．

[ステップ 1]

2.1 節の [ステップ 0] と同様，離散空間画像 $f(n_1, n_2)$ および $g(n_1, n_2)$ の二次元 DFT，$F(k_1, k_2)$，$G(k_1, k_2)$ を求める．

[ステップ 2]

$F(k_1, k_2)$，$G(k_1, k_2)$ の振幅スペクトルの対数をとった，対数振幅スペクトル $|F(k_1, k_2)|$，$|G(k_1, k_2)|$ を求める．

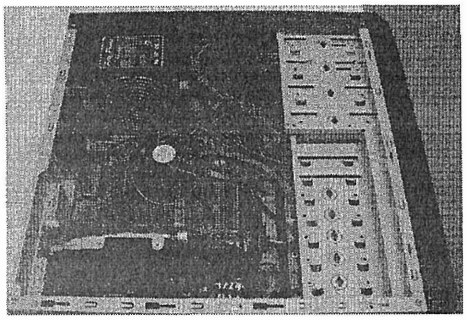

図 4　GPU による高速位相限定相関法計算システム

[ステップ 3]

対数振幅スペクトル $|F(k_1, k_2)|$ および $|G(k_1, k_2)|$ をそれぞれ極座標変換した画像 $F_p(m_1, m_2)$ および $G_p(m_1, m_2)$ を次式で求める．

$$\begin{aligned} |F_p(m_1, m_2)| &= |F(r_{m2}\cos\phi_{m1}, r_{m2}\sin\phi_{m1})| \\ |G_p(m_1, m_2)| &= |G(r_{m2}\cos\phi_{m1}, r_{m2}\sin\phi_{m1})| \end{aligned} \tag{9}$$

ここで，m_1，m_2 は極座標変換画像の座標であり，それぞれ $m_1 = -M, \cdots, M$，$m = -M, \cdots, M$ である．また，極座標変換の離散角度および離散半径は次式で表される．

$$\phi_{m_1} = \frac{\pi}{N} m_1, \quad r_{m_2} = m_2 + M \tag{10}$$

なお，極座標変換に必要な点が離散点間に存在する場合は双線形補間を行う．

[ステップ 4]

得られた極座標変換の画像 $F_p(m_1, m_2)$ および $G_p(m_1, m_2)$ に対して POC を適用し，回転量を推定する．

3. 実　　装

位相限定相関法の GPU 実装について説明する．

3.1 実装環境

今回は図 4 に示した Nvidia 社 GPU GeForce8800 を用いたシステムを構築し，図 3 のような画像レジストレーション処理が実現可能なところまで実装を完成させた．

まず，図 5 で Nvidia 社 GPU GeForce8800 のアーキテクチャを簡単に示す．図中の SP は Stream Processor の略である．これは並列に動作可能なスカラープロセッサで 8 個が SIMD 的に動作する．GeForce 8800 ではこの 8 個の StreamProcessor で 1 つのマルチプロセッサ (MP) を構成する．マルチプロセッサごとに 16KB 容量の共有メモリ (Shared Memory) を持つ．チップの外側には 512MB 容量のオンボードメモリ (GPU メモリ) を持つ．

CUDA では CPU 側をホスト，GPU 側をデバイスと呼ぶ．CUDA プログラムはホストプログラムと GPU カーネル関数から構成される．ホストプログラムは CPU 上で実行されるプログラムであり，通常の C 言語として実装し，GPU に対してデータ転送，GPU カーネル関数の起動を行う．GPU カーネル関数は GPU 上で実行されるプログラムであり，ホストプログ

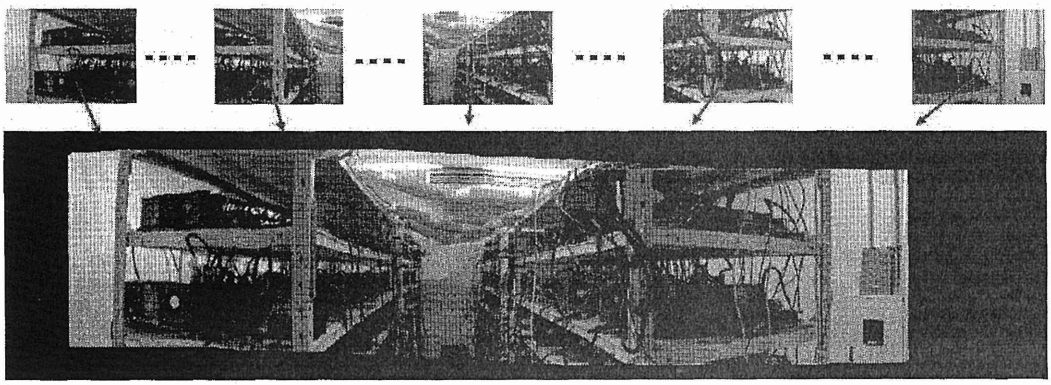

図 3　GPU による位相限定相関法を用いた画像合成例 (写真は長崎大学 GPU クラスター NGC190T). 連続した元画像 (上図) から一枚の合成画像 (下図) を自動生成した例.

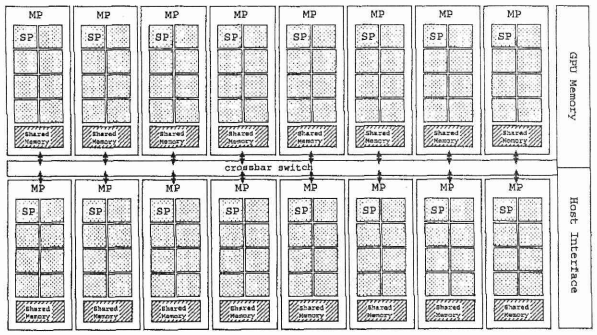

図 5　GeForce8800 のアーキテクチャ

ラムから起動される．CUDA プログラムは C 言語を拡張した言語仕様となっているが，カーネル関数内部では再帰処理，関数ポインタなどといった一部の機能に制限が存在する．

CUDA を用いた場合の基本的な計算処理の流れは次のようになる．

（1） 処理データをホストメモリから GPU メモリへ転送する

（2） ホストからデバイスへ処理を伝える

（3） デバイスの各コアが並列処理を実行する

（4） 処理結果を GPU メモリからホストメモリへ転送する

このように，CUDA では GPU へ処理データを明示的に転送する必要がある，GPU の処理は GPU メモリへ転送されたデータに対して行われる，カーネル関数は CPU が起動する，などが一般的な C 言語プログラミングと異なる点である (図 6).

図 7 は，CUDA のプログラミングモデルを示したものである．まず，ホストは GPU カーネル関数を呼び出す．デバイス側では，一つのカーネルに対して一つのグリッドが割り当てられる．一つのグリッドは，複数のブロックから構成され，一つのブロックは，複数のスレッドから構成される．スレッドとは処理を行う最小単位であり，ブロックとスレッドの数は，プログラマが任意に指定することができる．

3.2　POC の実装

POC の実装の際の留意点について，2.1 節の各ステップに沿って説明する．RIPOC の実装の際の留意点については，POC

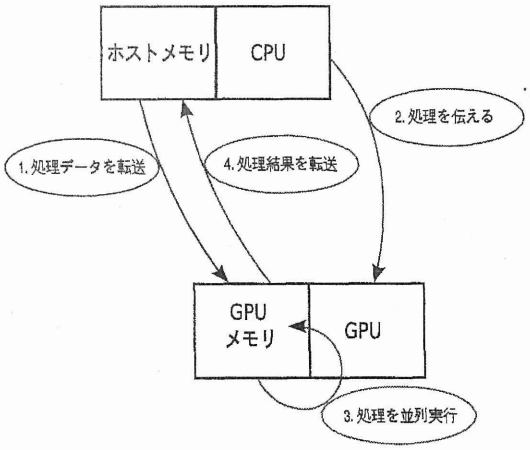

図 6　CUDA の処理の流れ

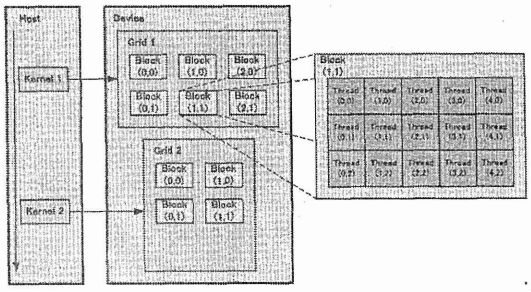

図 7　CUDA プログラミングモデル

の処理の各ステップとほぼ同様であるため，割愛する．

[ステップ 1]

二次元 DFT は CUDA のライブラリである CUFFT を使用した．ただし，式 3 において n_1，n_2 の範囲が $-M,\cdots,M$ で定義されているため，入力データを半分シフトさせ，範囲を $0,\cdots,2M$ とする必要がある．

[ステップ 2]

この時，同一ブロック内のスレッドは，空間距離の近いメモリを参照するように割り当てている．これは例えば，ブロック 1 で $0,\cdots,99$ 番地のメモリを使用する場合，ブロック 2 では $100,\cdots,199$ 番地のメモリを使用するように割り当てる，とい

うことである．

[ステップ 3]

位相限定相関関数 $r(n_1, n_2)$ は二次元 IDFT で求められるため，ここでも CUFFT を使用する．

[ステップ 4]

まず，位相限定相関のピクセル単位でのピーク座標を求める．次にサブピクセル単位でのピーク座標を求めるため，微小移動量 δ_1，δ_2 を $-1 \sim 1$ の範囲で変化させ，繰り返しフィッティングを行う．δ_1，δ_2 の組み合わせ毎に一つのスレッドを利用することで，高速にフィッティングを行うことができる．

3.3 コードレベルでの実装の詳細

```
 1 struct PocData {
 2   double min_dx,  min_dx2;
 3   double min_dy,  min_dy2;
 4   double score_min;
 5   double max;
 6   int peak_x,  peak_y;
 7 };

 8 PocData
 9 cal_poc_on_cpu (double *img1, double *img2, int X, int Y)
10 {
11   PocData result;

12   my_FFTW(img1,  fftw_out1);
13   my_FFTW(img2,  fftw_out2);

14   for(int i=0 ; i<Y*(X/2+1) ; ++i) {
15     fftw_out1_power[i] = log( sqrt( fftw_out1[i][0]*fftw_out1[i][0] +
16                         fftw_out1[i][1]*fftw_out1[i][1] ) );
17     fftw_out2_power[i] = log( sqrt( fftw_out2[i][0]*fftw_out2[i][0] +
18                         fftw_out2[i][1]*fftw_out2[i][1] ) );
19   }
20   xy2rs(fftw_out1_power,  polar_coordinates1);
21   xy2rs(fftw_out2_power,  polar_coordinates2);

22   my_poc(polar_coordinates1, polar_coordinates2, NULL, &result);

23   my_rot(img1,  result.min_dy/Y*M_PI,  img1);

24   my_poc(img1,  img2,  NULL,  &result);

25   return (result);
26 }
```

図 8　回転不変位相限定相関法 (RIPOC) 処理の C 言語プログラム．

まず，RIPOC 処理を簡単に説明するために，C 言語で書いた場合のコードを図 8 に示す．このコードは，

1a　FFT の実行 (12-13 行目) ... FFT

2a　振幅スペクトルの極座標変換 (14-21 行目) ... POL

3a　POC 処理 (22 行目) ... POC

4a　画像を回転させる (23 行目) ... ROT

5a　POC 処理 (24 行目) ... POC

という処理を行っており，CPU のみで動作するものである．my_FFTW 関数の内部では FFT 処理の他に，前処理としてのデータシフト操作も含まれる．

次に，RIPOC 処理を GPU で処理する場合の CUDA コードを図 9 に示す．このコードは，

1b　データをデバイスにコピーする (11-14 行目) ... CPY

2b　CUFFT の形式にあわせるため，データをシフトさせ，CUFFT の実行 (15-18 行目) ... FFT

```
 1 struct PocData {
 2   double min_dx,  min_dx2;
 3   double min_dy,  min_dy2;
 4   double score_min;
 5   double max;
 6   int peak_x,  peak_y;
 7 };

 8 PocData  cal_poc_on_gpu(float *img1, float *img2,  int X,  int Y)
 9 {
10   PocData  result;

11   cudaMemcpyAsync( d_img1,  img1, sizeof(float)*Y*X,
12             cudaMemcpyHostToDevice, my_stream);
13   cudaMemcpyAsync( d_img2,  img2, sizeof(float)*Y*X,
14             cudaMemcpyHostToDevice, my_stream);

15   my_shift_exe2<<<BLKNUM,THLNUM>>>(d_img1, d_shift1,
16                         d_img2, d_shift2, X, Y);
17   cufftExecR2C(fft_plan_r2c, d_shift1, d_ffted1);
18   cufftExecR2C(fft_plan_r2c, d_shift2, d_ffted2);

19   my_power_exe2<<<BLKNUM, THLNUM>>>(d_ffted1, d_pow1,
20                         d_ffted2, d_pow2, Y*(X/2+1));
21   my_xy2rs_exe2<<<BLKNUM, THLNUM>>>(d_pow1, d_rs1,
22                         d_pow2, d_rs2, X, Y);

23   my_poc_on_gpu( d_rs1, d_rs2, NULL, &result , X, Y);

24   my_rot_exe<<<BLKNUM, THLNUM>>>(d_img1, d_rs1,
25                         result.min_dy/Y*M_PI, X, Y);

26   my_poc_on_gpu( d_rs1, d_shift2, NULL, &result, X, Y);

27   return (result);
28 }
```

図 9　回転不変位相限定相関法 (RIPOC) 処理の CUDA プログラム．

3b　振幅スペクトルの極座標変換 (19-22 行目) ... POL

4b　POC 処理 (23 行目) ... POC

5b　画像を回転させる (24-25 行目) ... ROT

6b　POC 処理 (26 行目) ... POC

という処理を行っている．画像データの CPU メモリから GPU メモリへのデータ転送は処理 (3.3) のみで，その他の処理データ転送は必要がない実装になっている．処理 (3.3) から (3.3) では，処理 (3.3) で GPU メモリに保存された画像データに対して次々に異なる処理をほどこすように種々の GPU カーネル関数を起動している．

表 1　位相限定相関法の処理時間の内訳 (単位はミリ秒)．CPY，FFT，POL，POC，ROT はそれぞれ 3.3 章の処理リスト (1a) から (6b) に対応している．

	256x256		512x512		1024x1024	
	GPU	CPU	GPU	CPU	GPU	CPU
CPY	0.18	-	0.48	-	1.64	-
FFT	0.30	4.47	1.17	23.30	4.23	111.62
POL	0.37	8.02	1.78	32.75	6.26	134.45
POC	0.70	33.99	2.04	67.88	6.97	223.91
ROT	0.11	1.10	0.41	4.42	1.59	17.83
POC	0.70	34.20	2.04	64.63	6.96	224.37
Total	2.36	81.78	7.92	192.98	27.65	712.18

表 3.3 は先ほど説明した CPU と GPU での RIPOC 処理

表 2　処理速度比較

Function	Cinderella II @ 100MHz	Pentium III PC @ 500MHz	Core 2 Quad @ 2.4GHz	GeForce 8800GTS
2D-FFT	8.88ms	96ms	2.23ms	0.13ms
POC	27.15ms	501ms	82.7ms	2.7ms

の内訳を対応づけ，それらに要した計算時間 (ミリ秒) を示したものである．CPY，FFT，POL，POC，ROT はそれぞれ CPU 処理リスト (1a)〜(5a)，および GPU 処理 (1b)〜(6b) に対応している．3 種類の画像サイズについて CPU, GPU で要した RIPOC 処理の計算時間を表している．CPU は Intel Core2Quad Q6600 1 コアを用い汎用 FFT ライブラリである fftw(注1)ttp://www.fftw.org/ の SSE2 を有効にしたものを利用した，GPU は GeForce8800GTS を 1 台使用した場合の結果である．CPU，GPU どちらもほぼ同じような処理内容の内訳になっていることがわかる．ただし，GPU についてはホストメモリからデバイスメモリへの画像データの転送 (処理 CPY) が必要であるが，CPU の場合は必要がない．また，GPU についてはデバイスメモリからホストメモリへの処理結果の転送は 32 バイト程度の非常に小さい転送サイズのため処理 POC に含めている．また，RIPOC 処理で精度向上に必要な窓関数の適用処理については，これらの処理とは別な関数として実装した．

4. 実験結果

今回の実験では 256 × 256 画素の画像に対し POC 処理を行い，処理時間を文献 [5] の専用 LSI と比較した．この結果を表 2 に示す．

ただし，表 2 の Core 2 Quad と GeForce の POC の処理時間は，RIPOC により回転量を求め，さらに POC で平行量を求めるまでの時間を測定したものである．POC のみの処理時間は Core 2 Quad で 34ms，GeForce で 0.7ms である．サブピクセル精度の移動量を求める方法として 11 × 11 点でのフィッティングを行っている．

処理速度の計測には gettimeofday 関数を用いた．メモリ確保，メモリ解放の時間については，それぞれプログラムの起動後及び終了直前に 1 度のみ行えばよいので処理時間には含めていない．

5. 議　　論

今回の処理時間の測定方法では GPU による実装で専用 LSI の 10 倍程度の速度を実現することが確認できた．しかし比較に用いた専用 LSI のデータ [5] が非常に古いこと，サブピクセル精度での移動量を求める際の計算方法が異なっている可能性があることなどから信頼できる結果を得るには，より多くの条件を一致させた議論を行う必要がある．FPGA との比較については，現在のところ実機動作による測定データが非常少ないため満足な比較ができなかった．

また，今回の実装では GPU を 1 台のみ用いて行ったが，複数台の GPU が利用可能な PC による並列処理や，GPU クラスタによる並列処理への拡張も検討する余地がある．

さらに，今回の実装では FFT 処理部分に汎用 FFT ライブラリを用いているが，位相限定相関法の特徴を考慮すると，汎用ライブラリよりも効率的な FFT 処理を実装し，さらなる高速化が可能であると考えている．例えば，今回用いたように 2 つのデータに個別のフーリエ変換を行う方法よりも，一括してフーリエ変換を行う方法が有り，無駄なデータ転送と演算処理を削減することが可能である．逆フーリエ変換についても振幅の値が必ず 1 となることがあらかじめ判っているので，この特徴を利用した処理の簡略化が可能である．このような単体 GPU での更なる高速化も今後の課題である．

hogehoge

6. ま と め

GPU(Graphics Processing Unit) を用いた高速化手法を考案し，RIPOC 処理を Nvidia GeForce8800 GTS へ CUDA を用いて実装した．GPU 1 台当たりの処理時間として 256×256 ピクセルの画像で 2.36 秒，512×512 ピクセルの画像で 7.92 秒，1024×1024 ピクセルの画像で 27.65 秒という計算速度を確認した．また，この結果と過去の専用 LSI や FPGA を用いた場合の測定結果と比較することで GPU による RIPOC 処理は約 10 倍程度高速である可能性を示唆することができた．

文　　献

[1] 安居院猛，長尾智晴，”画像の処理と認識”，昭晃堂 (1992)

[2] C.D.Kuglin, D.C. Hines, ”The phase correlation image alignment method”,Proc.Int.Conf.Cybernetics and Society, pp.163-165, 1975.

[3] Q.S.Chen, M.Defrise, F.Deconinck,” Symmetric phase-only matched filtering of Fourier-Mellin transforms for image registration and recognition”,IEEE Trans.Pattern Anal.Mach.Intell.,vol.16,no.12,pp.1156-1168,Dec.1994.

[4] 青木孝文，伊藤康一，柴原琢磨，長嶋聖，”位相限定相関法に基づく高精度マシンビジョン - ピクセル分解能の壁を越える画像センシング技術を目指して-”，IEICE Fundamentals Review, Vol.1, No.1, pp.30–40, July 2007.

[5] 森川誠，勝亦敦，石井秀昭，”画像処理 LSI ”Cinderella II ”の開発”，株式会社　山武 Savemation Review 1999 年 8 月　発行号

[6] 佐久間健，飯野徹，伊藤康一，青木孝文 ”位相限定相関法を用いた指紋認証アルゴリズムの FPGA 実装”，2008 年電子情報通信学会総合大会通信講演論文集 pp.503

[7] 森原浩之，山本紘督，平井慎一，”回転不変位相限定相関法を基にしたビジョンアルゴリズムの FPGA 実装”，第 5 回システムインテグレーション部門学術講演会 (SI2004)，SY0013/04/0000-0180 be

[8] 佐々木慶文，瀧田健児，青木孝文，樋口龍雄，小林考次，”位相限定相関法に基づく高精度レジストレーション”，電子情報通信学会技術研究報告. DSP, ディジタル信号処理, 102(13), pp.49-54, 20020411, (ISSN 09135685) (社団法人電子情報通信学会)

[9] 長嶋聖，青木孝文，樋口龍雄，小林孝次，”位相限定相関法に基づくサブピクセル画像マッチングの高性能化”，計測自動制御学会東北支部　第 218 回研究集会 (2004.10.9) 資料番号 218-15

(注1)：h