Vol.2012-CVIM-182 No.4
2012/5/23

GPGPUによるFerns Descriptorの学習の高速化

北川　正理[1,a)]　清水　郁子[1,b)]

概要：画像の特徴抽出手法のひとつである Ferns Descriptor は，インプリメントが容易でありマッチングが高速な手法であるが，前処理として行う学習に非常に計算時間がかかることが知られている．そこで，本研究では，Ferns Descriptor の学習時間を GPGPU を利用して高速化する．このとき，メモリアクセスやデータ転送が遅いことが問題になる．そこで，使用メモリを削減し，並列化を効率よく行うために，キーポイント数，Ferns 数，クラスが含むパッチ数の 3 通りで並列化を行い，学習で扱うデータによって GPU 内の処理をモジュール化し，特徴量を圧縮してメモリの効率化を図る．

キーワード：特徴抽出，Ferns Descriptor，GPGPU，CUDA

GPGPU Accelerated Ferns Descriptor

Kitagawa Masamichi[1,a)]　Shimizu Ikuko[1,b)]

Abstract: Ferns descriptor is one of the feature extraction methods. It is easy to implement and its matching is very fast. However, the computational time for its learning is very high. In this paper, a method for fast learning of Ferns descriptor by GPGPU is proposed. Data transfer between GPU and CPU and memory access is very slow. To reduce the memory usage, three parallelization are tested by number of keypoints, by number of Ferns, and by number of patches. To reduce the data transfer between GPU and CPU by introducing the one dimensional indices for the image coordinate and controlling the device memory in GPU by dividing learning step of the Ferns descriptor into small steps.

Keywords: Feature extraction，Ferns Descriptor，GPGPU，CUDA

1. はじめに

コンピュータビジョンの分野において，画像からの特徴抽出は最も重要なタスクの一つであり，様々なアプリケーションがある．そのため，多くの手法が提案されており，Scale Invariant Feature Transform (SIFT)[4] や Speeded Up Robust Features (SURF)[1] などがよく使われている．

近年提案された Ferns Descriptor[3], [5], [6] も有力な手法の一つとして知られている．Ferns Descriptor は，特徴点周辺のパッチの見た目を統計的に学習し，マッチングを行う．非階層的な手法であり，インプリメントが容易でマッチングが高速であるという利点がある．しかし，学習に比較的多くの時間がかかるという欠点がある．

一方，graphics processing unit(GPU) をレンダリングだけではなく汎用計算に用いるためのライブラリが整備され，General Purpose Graphical Processing Unit (GPGPU) が一般的になってきた．画像からの特徴抽出でも GPGPU を用いた高速化が行われるようになり，例えば GPU SIFT[8], GPU SURF [2]，CUDA SURF [7] などが提案されている．

本稿では，Ferns Descriptor の学習ステップを GPGPU を用いて高速化する手法について述べる．Ferns Descriptor の学習ステップは，多数の互いに依存しない独立な処理により構成されているため，GPGPU による並列計算で高速化が見込める．GPU を用いた計算では，GPU のメモリ量は CPU のメモリ量に比べて小さいこと，GPU と CPU 間のデータ転送スピードは非常に遅いことが実装上問題にな

[1] 東京農工大学大学院
Tokyo Univ. of Agri. and Tech., Koganei, Tokyo 184–8588, Japan
[a)] masamichi.stabil.bass.linie@gmail.com
[b)] ikuko@cc.tuat.ac.jp

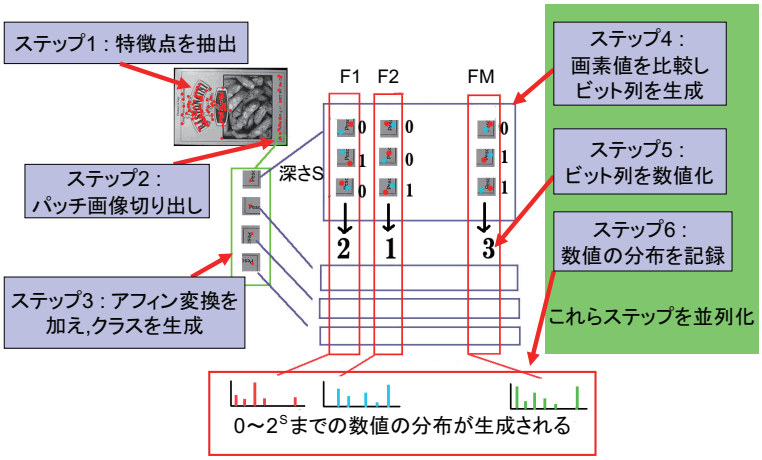

図 1 Ferns Descriptor の学習ステップ.
Fig. 1 Learning step of the Ferns descriptor.

る．そこで本手法では，画像座標を 1 次元のインデックスで表すことにより GPU と CPU 間でのデータ転送量を削減し，学習ステップの処理をモジュール化し，メモリ管理を適切に行う．これにより，GPU でのメモリ使用量が削減され，データ転送時間が減少するため，全体の処理時間を削減する．

2. Ferns Descriptor の学習ステップ

本節では，Ferns Descriptor[3], [5], [6] の学習ステップについて説明する．あらかじめ学習する画像（参照される画像）のことを「モデル画像」と呼ぶことにする．

モデル画像の学習 (図 1) のために，まずヘッセ行列に基づきコーナー検出を行うことにより画像中の特徴点 (key points) を求める．次に，特徴点の周辺の小領域（パッチ）を抽出し，各パッチに対して様々なアフィン変換を施す．様々なアフィン変換を施して得られる画像は，仮想的に生成した特徴点周辺の見た目に相当しており，クラスと呼ばれる．特徴点数を N とし，T 種類のアフィン変換を施すとする．つまり，一つのクラスには T 個の変換されたパッチ画像が含まれ，全体では，$N \times T$ 個のパッチが生成される． このとき，T 種類のアフィン変換は，N 個の特徴点全てに対して共通のものを用いる．一般に，N は約 100〜500，T は約 10〜300 程度が用いられる．

次に，各クラスの各パッチに対し，M 個の異なる 2 つの画素の組に対し，画素値を比較する．ただし，2 画素の組の位置は全てのクラスに対して共通であるが，その場所は任意である．もし点 A(図 1 の赤の点) の画素値が点 B (図 1 の水色の点) の画素値より小さければ 0，そうでなければ 1 とし，M 個のビット列が生成される．この $M \times S$ 回の比較は，Ferns Descriptor の重要な処理である．M は Ferns 数と呼び，S を Fern の深さと呼ぶ．

このようにして得られた M 桁のビット列を S ビットの数値として表す．このようにして，$N \times M \times T$ 個 1bit$S$ 桁のビット列が，$NtimesT \times M$ 個の Sbit の数値に変換される．一般に，M は 10〜300 程度であり，S は 64 またはそれ以下が選ばれる．

3. Ferns Descriptor の学習ステップの GPGPU による高速化

本節では，GPGPU による Ferns Descriptor の学習ステップの高速化について説明する．本手法では，GPGPU のプログラミングには CUDA を用いて実装した．CUDA ではスレッド数は使用するデバイスにより，512 か 1024 と定められている．

本手法では，図 1 の緑色の四角で示した部分を並列化する．具体的には，ビット列の生成， ビット列の数値化，数値の記録である．これらのステップでは，N, T, M, および S は独立のループであるが，CUDA のスレッド数が限られているため全てを同時に並列化することはできない．なるべく多くのスレッドが生成されるように並列化するためにパラメタを選ぶが，これらの 4 つのパラメタの中で，S は他の 3 つよりも小さいので，N, T, M の 3 通りの並列化を行う．特徴点数 N による並列化は図 2，クラス内パッチ数 T による並列化は図 3，Ferns 数 M による並列化は図 4 に示す．

ここで，CUDA にはスレッドのほかに関数をコピーしてループのインデックスをあらかじめ演算しておけるようにブロックという処理系が存在し，こちらは上限が 65565 であることに着目する．ブロックでは，2 次元でのインデックス指定が可能であるので N, T, M のどれかで並列化を行なった場合には残りの 2 つのパラメータでブロックを生成できる．これにより関数のループ処理を軽減することができるので, 本研究のプログラムではループ処理は深さ S に対してのみ行なっている．

GPGPU 導入の際の最も大きな問題は，CPU と GPU の間はバスで接続されており，データ転送速度が非常に遅い

ことである．特に CPU から GPU への転送は，GPU から CPU の転送に比較して更に遅い．さらに，GPU のメモリサイズは一般に CPU のメモリサイズに比較して小さいことも注意しなければならない．そのため，高速化には適切なメモリ制御が不可欠である．

そこで，本手法では，ビット列を生成する段階で必要な画素値比較用の座標を GPU で生成することにより CPU から GPU へのデータ転送を削減する．このとき，クラス内パッチのデータを全て 1 次元に並べることで通常は 2 次元のインデックスを持って行われる座標の指定を 1 次元のインデックスで行えるようにする．そしてこれら座標の数値を GPU 側で計算してホストと共有する．画素値比較用の点座標はマッチングの段階でも利用されるため，CPU 側と共有する必要があるが，GPU 側への転送の遅さから，GPU 上で点座標を生成し，CPU 側へ転送するもしくは CPU で全く同じ点座標を生成する方が転送時間の削減につながる．

さらに，GPU のメモリ領域を有効に利用するために処理をモジュール化する．ビット生成ステップ, 数値変換ステップ, 数値分布記録ステップに処理を分割し，これらの処理のあいだにメモリ領域の開放と確保を適切に行う．ビット列を生成するステップではすべてのクラス内パッチ画像, 画素値比較用の点座標データ， ビット列を保持するためのデータ領域が必要になるが, ビット列を数値に変換するステップではパッチ画像と比較用の点座標データは必要なくなるので領域を開放する．その上で変換したあとの数値を記録するステップでは，データ領域を確保する．同様に数値の分布を記録する段階ではビット列データは必要なくなるので開放し， 数値分布記録用のデータ領域を確保する．このようにメモリ管理を行うことでメモリ領域を有効に利用することができる．

4. 実験

本節では，提案手法の有効性を示すための実験結果を示す．実験では，2 種類の CPU と 4 種類の GPU を用い，それぞれのプロセッサを用いた場合の計算速度を比較した．また，3 種類の並列化（特徴点数による並列化，クラス内パッチ数による並列化，Ferns 数による並列化）の比較を行った．さらに，画素値比較用座標を 1 次元のインデックスで表現し, かつ GPU 側での点座標生成を行った提案手法と，これを適用しない手法の計算時間の比較を行った．

実験条件を下記に示す．

- CPU: Intel Core i5 2400, Athlon II ×2　260,
- GPU: Geforce 210, Geforce GTS 450, Geforce GTX 295, Quadro 600,
- CPU のメモリ量: 4GB DDR3,
- OS: Windows7　professional 32bit,
- CUDA のバージョン: CUDA SDK 2.3 32bit,

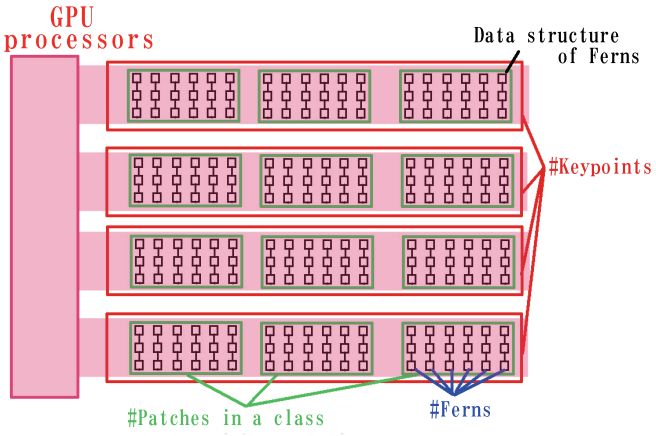

図 2 特徴点数による並列化.
Fig. 2 Parallelization by the number of keypoints.

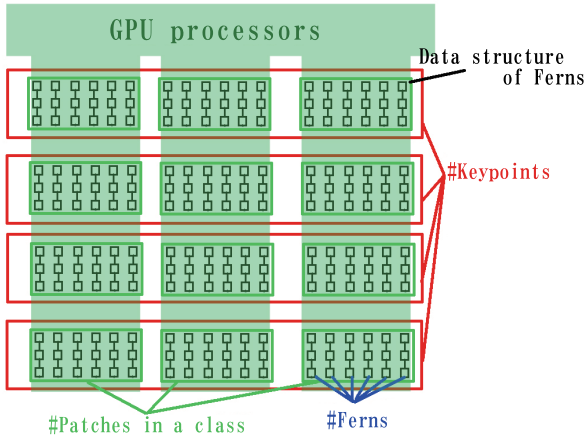

図 3 クラス内パッチ数による並列化.
Fig. 3 Parallelization by the number of patches in a class.

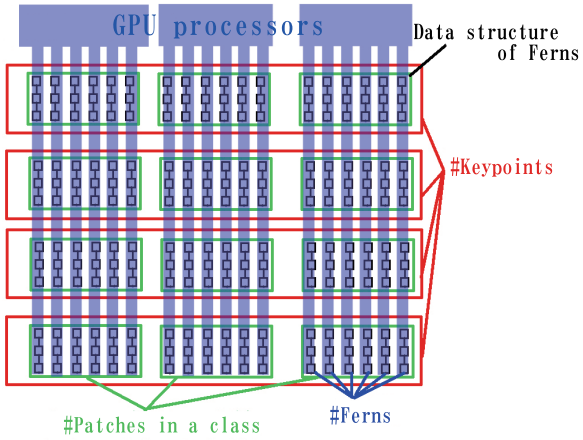

図 4 Ferns 数による並列化.
Fig. 4 Parallelization by the number of Ferns.

表 1 GPGPU を用いた提案手法による計算時間の内訳と CPU による計算時間の比較.
Table 1 Details of the computational time by our implementation using GPGPU comparing with the implementation using CPU.

クラス内パッチ数	50	100	150	200	250
総計算時間 (Geforce GTS 450) [ms]	188.4	264.9	334.0	405.4	483.3
GPU での処理時間 (Geforce GTS 450) [ms]	0.047	0.055	0.044	0.047	0.051
総計算時間に対する GPU での処理時間の割合 [%]	0.029	0.021	0.013	0.012	0.011
総計算時間 (Geforce 210) [ms]	851.9	1252.8	1688.4	2091.5	2508.3
GPU での処理時間 (Geforce 210) [ms]	0.087	0.081	0.076	0.097	0.131
総計算時間に対する GPU での処理時間の割合 [%]	0.010	0.006	0.005	0.005	0.005
CPU での処理時間 (Core i5 2400) [ms]	503.5	1054.7	1625.0	2217.9	2775.7

- コンパイラ: Visual studio 2008 32bit.

実験結果は図 5，図 6，図 7 およびに示す.

図 5，図 6，図 7 に示した実験では，T を $T = 60, 90, 120, 150, 180$ の 5 種類に変化させ，N, M, S はそれぞれ $N = 100$，$M = 50$，$S = 64$ に固定した．表 1 に示した実験では，T を $T = 50, 100, 150, 200, 250$ の 5 種類に変化させ，N, M, S は $N = 100$，$M = 50$，$S = 64$ に固定した.

図 5 は，2 つの CPU および 4 つの GPU による計算時間を比較した結果である．GPU により並列演算を行うことで，高速に Ferns Descriptor の学習が行うことができたことがわかる．特に Geforce GTS 450 と Quadro 600 は Core i5 よりも安価であるにもかかわらずパフォーマンスが上回っている.

図 6 には，3 種類の異なる並列化（特徴点数による並列化，クラス内パッチ数による並列化，Ferns 数による並列化）の結果を示している．これらの実験は，Geforce GTS 450 により実行した．クラス内パッチ数 T を増加させるにつれ，クラス内パッチ数による並列化が有効である．また，特徴点数や Ferns 数を増加させて実験すると，特徴点数や Ferns 数による並列化が有効になることを確認している．これらの結果から，できるだけ多いパラメタでスレッドを生成した場合に最も効率よく演算が行えていることがわかった.

図 7 は，画素値比較用座標を 1 次元インデックスで表現し，かつ画素値比較用の点座標生成を GPU 側で行った提案手法による結果 (“advanced”) とこれを行わなかった場合の結果 (“dataimport”) を比較している．本手法により高速化が行われていることが確認できた.

表 1 は，提案手法にかかったデータ転送時間と実際に GPU において計算を行った時間を示している．トータルの計算時間のうち，99%以上の時間がデータ転送に費やされていることが確認された．計算時間の削減には，GPU 側でクラスの生成を行うためのさらなる工夫が必要になると思われる．具体的には，特徴量抽出の演算並列化するだけでなく，クラスの生成も並列化し，大量のクラス内パッチ画像の代わりにモデル画像を転送することでデータの転送量を減らすことが考えられる.

5. 結論

本論文では, Ferns Descriptor の学習ステップに GPGPU を導入することにより高速化を行った．GPGPU により並列計算を行うことによって，Ferns Descriptor の学習ステップの計算時間は，CPU で実行する場合の計算時間に比べ，大幅に削減された.

本手法では，特徴点数による並列化，クラス内パッチ数による並列化，Ferns 数による並列化の 3 種類の異なる並列化を実装した．CPU と GPU の間のデータ転送時間が非常に大きいことを考慮し，画素値比較用の画像座標を 1 次元のインデックスにより表現することで，データ転送量を削減した．また，処理を細かいモジュールに分割し，各ステップで必要なデータ領域のみを確保し，適切にメモリの開放を行うことによって高速化を行った.

実験結果により，本実装による GPGPU を用いた並列計算が CPU に比べて非常に高速であることが確認された．特に，Geforce GTS 450，Quadro 600 では，低価格であるにも関わらずハイスペックな CPU である Core i5 を上回る性能を示した．しかしながら，計算時間の内訳を精査したところ，ほとんどの時間がデータ転送にかかっていることが分かった．メモリ制御により更なる高速化を行うことが今後の課題である.

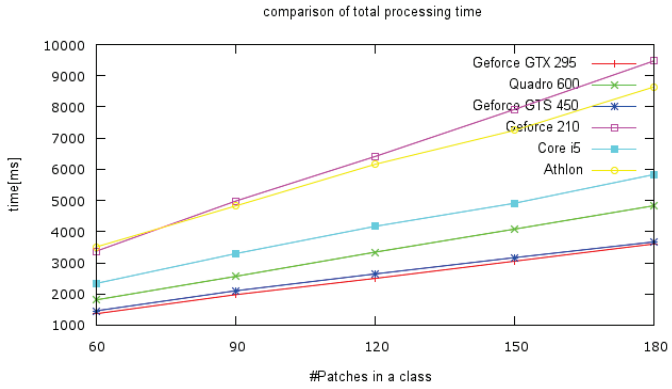

図 5 2 つ CPU および 4 つの GPU による計算時間の比較.

Fig. 5 Comparison of computational time by two CPUs and four GPUs.

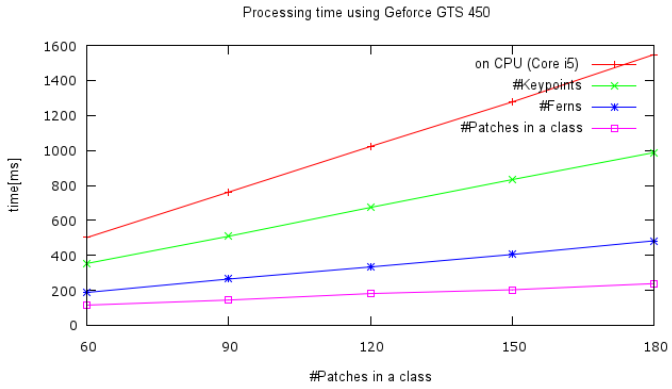

図 6 3 つの並列化（特徴点数による並列化，Ferns 数による並列化，クラス内パッチ数による並列化）の計算時間の比較.

Fig. 6 Comparison of computational time of three different parallelizations: by the number of keypoints, by the number of Ferns, and by the number of patches in a class.

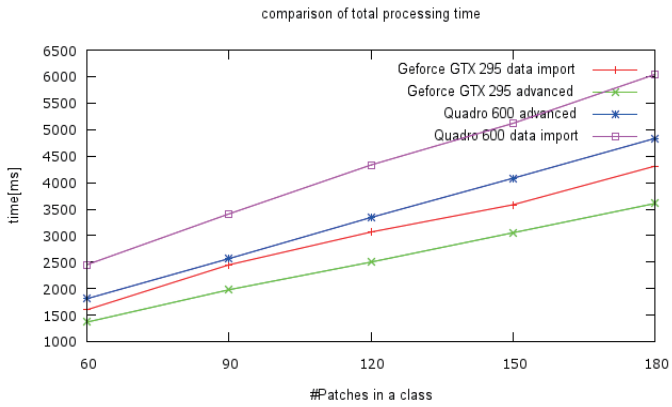

図 7 提案手法 (“advanced”) による計算時間とメモリ制御なしの手法 (“data import”) による計算時間の比較.

Fig. 7 Results by our algorithm (“advanced”) and by an algorithm without our memory control(“data import”).

参考文献

[1] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool. SURF: Speeded up robust features. *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[2] N. Cornelis, L. V. Gool, and K. Leuven. Fast scale invariant feature detection and matching on programmable graphics hardware. In *Computer Vision and Pattern Recognition Workshop*, pages 1–8, 2008.

[3] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1465–1479, 2006.

[4] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[5] P. F. M. Calonder, V. Lepetit. Keypoint signatures for fast learning and recognition. In *Proc. of European Conference on Computer Vision*, pages 58–71, 2008.

[6] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast keypoint recognition using random ferns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(3):448–461, 2010.

[7] A. Schulz, F. Jung, S. Hartte, D. Trick, C. Wojek, K. Schindler, J. Ackermann, and M. Goesele. Cuda surf - a real-time implementation for surf. `http://www.d2.mpi-inf.mpg.de/surf?q=surf`, 2010.

[8] C. Wu. SiftGPU: A GPU implementation of scale invariant feature transform (SIFT). `http://cs.unc.edu/~ccwu/siftgpu`, 2007.

[9] Jason Sanders, Edward Landrot:
”CUDA by Example 汎用 GPU プログラミング入門”,
インプレスジャパン (2011-2-11)

[10] 岡田賢治, 小山田耕二:
”CUDA 高速 GPU プログラミング入門”,
秀和システム (2010-3-25)