H-006

画像識別処理のための効率的な CUDA 実装に関する一考察

A study on efficient implementation of image recognition algorithm in CUDA

大西 茂彦† 服部 静枝† 細谷 英一† 青木 孝† 小野澤 晃†
Shigehiko Onishi Shizue Hattori Eiichi Hosoya Takashi Aoki Akira Onozawa

１．はじめに

画像特徴量に HOG (Histogram of Oriented Gradients)を用いた画像識別処理の例として，人物検出は良く知られた一つである．HOG は画像の輝度勾配をヒストグラム化したもので，その算出に相応の計算コストを要する．そのため，HOG を用いた人物検出処理の実装に関して GPGPU による効率化を図った例が報告されている[1]．本稿でも HOG を用いた Real Adaboost による人物検出アルゴリズムに関し，GPGPU による実装の効率化について報告する．特に，複数の異なったアーキテクチャの GPU を用いて，プログラム上の制約と実装手法の関係に注目している．

２．人物検出アルゴリズム

本稿の人物検出アルゴリズムは，HOG 特徴量の算出部と Real Adaboost による人物検出部の 2 つから成る[2].

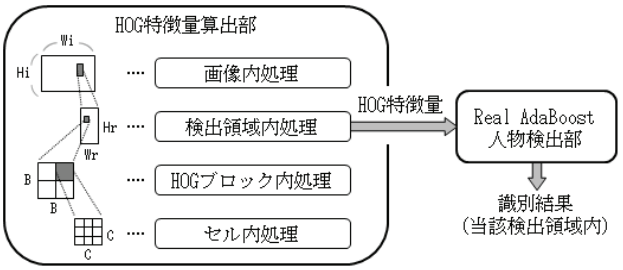

図 1　人物検出処理の構成の概略

2.1　HOG 特徴量算出部

HOG 特徴量算出部は 4 階層の処理から成る．

（１）　最下層のセル内処理は，C×C 個のピクセルを 1 セルとし，セル内の各ピクセル *p(u,v)*につき(2.1)及び(2.2)式で輝度の勾配強度 *M(u,v)*と勾配方向 *H(u,v)*を求め，方向毎に強度を積算したヒストグラムを作成する．ヒストグラムのビン数 Dc は方向の量子化レベル数で決まる．

$$M(u,v) = \sqrt{U(u,v)^2 + V(u,v)^2} \qquad (2.1)$$

$$H(u,v) = \tan^{-1}(V(u,v)/U(u,v)) \qquad (2.2)$$

$$U(u,v) = p(u+1,v) - p(u-1,v) \qquad (2.3)$$

$$V(u,v) = p(u,v+1) - p(u,v-1) \qquad (2.4)$$

（２）　第 2 層の HOG ブロック内処理では，B×B 個のセルを 1HOG ブロックとし，HOG ブロック内の B×B 個のヒストグラムのノルムの総和を 1 に正規化する．

（３）　第 3 層の検出領域内処理では，検出対象とする Wr×Hr ピクセルの部分画像領域（検出領域）内で HOG ブロックを１セルずつ縦横に移動させつつ（２）を行う．得られた正規化ヒストグラム全てを１次元ベクトルの形にし HOG 特徴量ベクトルとして後続の Real AdaBoost 検出処理に渡す．検出領域内のセル数 Nc は(Wr/C)×(Hr/C)個，HOG ブロック数 Nb は(Wr/C-B+1)×(Hr/C-B+1)である．

† NTT マイクロシステムインテグレーション研究所,
NTT Microsystem Integration Laboratories

HOG 特徴量ベクトルの次数 Dh は Dc×(B×B)×Nb となる．

（４）　最上層の画像内処理では，Wi×Hi ピクセルの入力画像全体にわたりセル単位で検出領域を縦横に移動させつつ（３）を行う．画像１枚当たりの HOG 特徴量ベクトルの個数 Nh は(Wi-Wr)/C×(Hi-Hr)/C 個となる．

2.2　Real AdaBoost による人物検出部

Real Adaboost による人物検出部は，M 個の弱識別器の出力を統合して最終的な識別関数を構成し，検出領域内の画像が人物か否かの 2 クラス識別を行う．各弱識別器は，それぞれ検出領域内の HOG 特徴量ベクトル中の特定の成分のみを入力とし，その値に応じて事前の学習結果を参照し出力を算出する．そして，それらの出力の総和を最終的な識別結果とする． 事前の学習結果とは，ポジティブサンプル（人物画像）4900 枚に対して得た確率密度値のテーブル Wp 及び入力とすべき成分番号のテーブル ip と，同数のネガティブサンプル（非人物画像）から得た同様のテーブル Wm 及び im である．これらから，i 番目の弱識別器の出力 *h(i)*を(2.5)式で算出する．

$$h(i) = 0.5 \times ln\{(vp(i) + e)/(vm(i) + e)\} \qquad (2.5)$$

vp(i)は ip(i)が示す番号の HOG 値を量子化しこれをインデクスとして Wp を参照した値であり，vm(i)は同様に im(i)が示す番号の HOG 値で参照した Wm のエントリの値である．なお，e は定数 0.001 とした．

3．実装

3.1　アルゴリズムの並列性

2.1 節の HOG 特徴量抽出部に関し，本検討ではセルサイズ C を 3，HOG ブロックサイズ B を 2 とし，勾配方向を 0°から 180°まで 20°毎に量子化して１セル当たりのヒストグラム次数 Dc を 9 とした．さらに，検出領域サイズ(Wr,Hr)を(18,36)とし，入力画像サイズ(Wi,Hi)を(576,432)とする．この条件の下で，2.1 節の HOG 特徴量抽出部の各処理について並列に扱い得る構成単位の数を考えると，セル内処理では 9 個のピクセル或いはヒストグラムのビン 9 個，HOG ブロック内処理では 4 個のセル，検出領域内処理では 72 個のセル又は 55 個の HOG ブロック，そして画像内処理では 24552 個の HOG 特徴量ベクトルとなる．

次に，2.2 節の Real AdaBoost 検出部について並列に取り扱い得る構成単位を考えると，M 個の弱識別器を単位とするのが自然である．本検討では M を 500 とした．

3.2　GPU ごとの実装

各処理の並列性の規模を把握した上で，GPU ごとの実装の最適化を検討する．実装は，GPGPU 分野で一般的な CUDA 技術を利用した．CUDA プログラミングでは，並列実行の処理単位を記述したカーネル関数を設計し，その呼び出し時に並列実行数をパラメータで指定する．パラメータにはグリッド内ブロック数とブロック内スレッド数の 2 つが指定でき，階層的な並列化が可能である．ブロック数とスレッド数を大きく設定するほど実行速度の向上が期待

されるが，メモリアクセスの遅延等も影響してその傾向は単純ではない．特に，グローバルメモリへのアクセス遅延が全体性能のボトルネックとなりがちで，GPU に備わる種々の特殊なメモリを用途とサイズに応じて使い分ける必要がある．本検討では，全アルゴリズムを GPU 内の高速な共有メモリ上で実装し，リードキャッシュを備えたテクスチャメモリやコンスタントメモリから入力画像や Real AdaBoost 用の学習データを読むこととした．さらに，浮動小数点演算は全て単精度で行い，数値関数（逆正接 atan2f()等）の計算には GPU 内の高速ルーチンを利用した．

以下では，2 種類の GPU について，スレッド数と共有メモリ量の違いを考慮した実装を検討する．

（１） 8800GT（NVIDIA 製 GPU）では，スレッド数の上限は 512，共有メモリ量の上限は 16kB に制限される．この制約の下で HOG 特徴量算出部の実装を考える．並列規模の大きい処理を並列化すれば大きな高速化が期待できるので，画像内処理（規模 24552）はグリッド内ブロックを割り当てて並列化する．そして，次に規模の大きい検出領域内処理（規模 72）をブロック内スレッドにより並列化する．HOG ブロック内処理（規模 4）も併せればさらに並列度が増す（スレッド 72×4 本）が，正規化処理は並列展開しにくく，性能はほとんど向上しない．また，使用した共有メモリの量は単精度型変数 2772 個分 11088 バイトとなり，制限内に収まった．次に，Real AdaBoost 人物検出部の実装を考える．この処理は HOG 特徴量算出部の検出領域内処理に続いて実行されるので，それと同数の 72 本のスレッドを割り当て，500 個の弱識別器を 72 個ずつひとまとめにして並列化する．500 個の弱識別器出力の総和は，既知の効率的な方法[3]を 72 ごとの部分和を求める形に変更して適用した．使用した共有メモリの量は，弱識別器の出力を一時保持する 500 個の単精度型変数 2000 バイトであるが，共有メモリの再利用により実質的な増分はない．

（２） Fermi アーキテクチャとして知られる GTS450 と GTX580（いずれも NVIDIA 製 GPU）では，スレッド数の上限が 1024，共有メモリ量の上限が 48kB に拡大され，より規模の大きな並列演算を実装し得る．前者は廉価版，後者は演算リソースが充実したハイエンド版という違いがあるが，現行 GPU による性能の把握のため双方を使用した．

（１）の場合との差異に留意しつつ HOG 特徴量算出部の実装を考えると，（１）では検出領域内処理のみにスレッド 72 本を割り当てるにとどまったが，この場合はさらに，規模 9 のセル内処理も並列展開できる．全スレッド数は 72×9 の 648 で制限内に収まる．使用する共有メモリの量は（１）の場合より整数型変数 648 個分 2592 バイトだけ増加するに留まった．この増加は，セル内処理が並列化された結果，ヒストグラム作成時に 9 ピクセル分の勾配方向値を保持しておく必要が生じたことによる．この増加でも，共有メモリの使用量は制限内に十分収まる．

Real AdaBoost 人物検出部の実装は（１）の場合より簡潔で，648 本のスレッドを 500 個の弱識別器の処理に 1 つずつ割り当てられる．使用する共有メモリの量は（１）の場合と同じく実質的に増加しない．

4．実験及び結果

GPU 毎に PC を 1 台ずつ用意し，Linux(Ubuntu 10.04 LTS)上で CUDA3.2 開発環境等の実験環境を整えた．また，実験用の CUDA プログラムとして 2 種類のものを用意した．一方は，HOG 特徴量算出部と Real AdaBoost 人物検出部の双方を１つの CUDA カーネル関数に納めたもの（結合版プログラム）で，もう一方は，それぞれを別の CUDA カーネル関数に収めてグローバルメモリを介しデータを受け渡すもの（分離版プログラム）である．後者は，共有メモリ量がさらに制限される場合や異なる人物検出部（SVM 等）を併用する場合の性能変化を推測するために用意した．

実行速度の測定には GPU 内のタイマー機能を利用し，ホスト側(CPU 側)が入力画像をテクスチャメモリに置いてからカーネル関数の実行が終了し判定値をホスト側に返すまでを測定した．

表 1 に各 GPU についての実行時間の測定結果を示す．結合版プログラムは分離版より約 30%以上高速であり，グローバルメモリアクセスによる速度劣化が大きいことがわかる．総じて，スレッド数を多く利用するものが高速であるが， GTS450 での分離版プログラムについてはスレッド数 72（50.0ms）の方が 648 の場合（58.7ms）より高速であるという例外的な結果となった．プロファイラで解析した結果，分離版 648 スレッドのプログラムは GTS450 上での実行の際に共有メモリアクセスでのバンクコンフリクトを多発し，これが速度劣化の原因であることが判明した．この現象は，スレッド番号と共有メモリアドレス値が 2 のべき乗倍（1 を除く）の関係にある場合に主に発生する．共有メモリ上の HOG 特徴量をグローバルメモリに書き出す際にこの問題が生じていたので，HOG 特徴量のメモリ上でのフォーマットをスレッド番号順とする改良を施した．改良後の測定結果も表 1 に示す．改良後の分離版 648 スレッドの結果(46.0ms)は 72 スレッドの場合（50.0ms）より高速となり，予測どおりの傾向を示すものとなった．

表 1　実行時間の測定結果

プログラム種類	8800GT	GTS450	GTX580
結合版　72 スレッド	38.1ms	36.7ms	8.63ms
結合版 648 スレッド	—	29.3ms	7.79ms
分離版　72 スレッド	49.9ms	50.0ms	15.2ms
分離版 648 スレッド	—	58.7ms	14.9ms
改良後の 分離版 648 スレッド	—	46.0ms	12.0ms

5．まとめ

人物検出アルゴリズムを CUDA により実装した結果を報告した．576×432 ピクセルの入力画像に対する処理時間は最も高速なケースで約 8ms となり，同サイズの動画像への適用においてスループット 33fps 以上を実現できる可能性が見出せた．今後はマルチ GPU での実装手法等も検討していく予定である．

参考文献

[1] Victor Prisacariu and Ian Reid: fastHOG – a real-time GPU implementation of HOG, Technical Report 2310/09, Department of Engineering Science, Oxford University, 2009

[2] 藤吉ほか: コンピュータビジョン最先端ガイド 2, アドコム・メディア, 2010

[3] 青木ほか: はじめての CUDA プログラミング, 工学社, 2009