Vol.2011-MPS-82 No.21
2011/3/8

GPU を用いた複数顔検出の高速化に関する一手法

久保　友理恵[†1]　　高田　雅美[†1]　　城　和貴[†1]

本稿では，複数の人物の顔を高速に検出するプログラム開発のために，GPU(Graphic Processing Unit) を用いた高速化の一手法を検討する．GPU として，NVIDIA 社の tesla C1060 を採用する．tesla C1060 では，GPGPU(General Purpose Computing on Graphic Processing Unit) 用開発環境である CUDA が使用可能である．顔検出には高速に検出可能な Haar-Like 特徴を用いた AdaBoost ベースの識別器を利用する．検討手法では，入力画像上をランダムに選んだ複数の位置から走査を開始し，Haar-Like 特徴演算を GPU で行うことによって，CPU での実装より高速な処理を可能にする．

A Multiple Faces Detection Method using GPU

Yurie Kubo,[†1] Masami Takata[†1] and Kazuki Joe[†1]

In this paper, we present some optimization techniques for multiple faces detection using GPU. As the target platform, we use tesla C1060 which is widely know as a high-end GPU. We use CUDA as the development environment for GPGPU. To detect face, we use a classifier based on AdaBoost with using Haar-Like features which can detect the face positions very quickly. In this case, we perform the Haar-Like feature calculation on tesla C1060 to demonstrate a faster implementation rather than CPUs.

1. はじめに

近年，犯罪発生率の急増に伴い，監視カメラによるセキュリティシステムの重要性がますます高まっている．最近の計算リソースの増大に伴い，顔認識に関する研究が急速に進んでいる．バイオメトリクスやセンサ等による個人識別は，その場所に居る人々の協力なしに人物を特定することが非常に困難である．しかし，顔認識は人々の協力なしに人物を特定することが可能な技術である．そのため、駅構内や店内など，人物が自由に行き交える場所でのセキュリティーが求められる中で，顔認識は非常に重要な技術となる．

現在では監視カメラは，どこに行っても見かけるようになってきている．しかし，現在の監視カメラを使ったシステムには 2 つの問題がある．1 つ目の問題点は，1 度に大人数の顔を認識することができない点である．特定の指定された位置の人の顔を認識することは可能だが，不特定多数の人の顔を認識することは不可能である．2 つ目の問題点は，リアルタイム処理ができない点である．監視カメラではリアルタイム性が必要であり，これによって様々な用途に用いることができる．したがって，複数の人物の顔認識のリアルタイム処理には，多くのニーズがあると思われる．そのため，まずは，複数の人物の顔検出の高速化に着目し，GPU を用いてそれを実行する．

以下，2 章では Haar-Like 特徴を用いた顔検出について述べる．3 章では，GPU での高速化にあたっての留意点について述べ，4 章では，実際に Haar-Like 特徴を用いた複数の人物の顔検出の実装と評価について，5 章では，まとめを述べる．

2. Haar-Like 特徴を用いた顔検出

Haar-Like 特徴という弱識別器に着目する [1][2]．Haar-Like 特徴のアルゴリズムは非常に簡単であるが，検知の性能はとても優れている．また，画像サイズが 320 × 240 程度であれば，2GHz の CPU で 10〜15fps で検知可能であるとされており，非常に高速である．学習アルゴリズムとして，AdaBoost アルゴリズムを用いる [3]．AdaBoost は，識別能力

†1 奈良女子大学人間文化研究科
Graduate School of Humanities and Sciences, Nara Womens's University

Vol.2011-MPS-82 No.21
2011/3/8

があまり高くない識別器をたくさん繋ぐことで，強い識別器を作るという考え方の学習アルゴリズムである．強識別器は，弱識別器にその重要度にあわせて重みをつけて組み合わせたものである．Haar-Like 特徴を弱識別器とし，AdaBoost のアルゴリズムで学習させ，強識別器を作成する． また，Haar-Like 特徴の計算は，Integral Image という手法を用いると高速に計算できる [4]．Integral Image は，任意の矩形領域の総和を高速に計算する手法である．あらかじめ，画像の特定領域の画素値の総和を計算しておくことで Haar-Like 特徴を用いたマッチングも高速になる．

Haar-Like 特徴とは，図 1 のような矩形領域の白黒パターンのことをいう．Haar-Like 特徴は，矩形特徴の黒領域の輝度値の合計から，矩形特徴の白領域の輝度値の合計を引いた値を特徴量として用いる．図 2 に，Haar-Like 特徴を顔画像の上においた場合の例を挙げる．識別率は，1 つの Haar-Like 特徴に対する識別率は高くないが，カスケードすることで高い識別率を得る．カスケードとは，強識別器を複数連結することである．認識対象の画像のほとんどは，認識対象外である．その大部分を処理の早い段階で切り捨てていくことで高速に処理することができる．図 3 の様に強識別器をつなぎ，識別率の低い識別器から始め，対象外であると識別された画像を除き，徐々に識別率の高い識別器にする．検出器の学習は 1 番目の強識別器から順に進める．各識別器には，目標とする認識率と誤識別率の許容範囲を設定し，それを達成するまで Haar-Like 特徴を作成・追加しながら学習を行う．

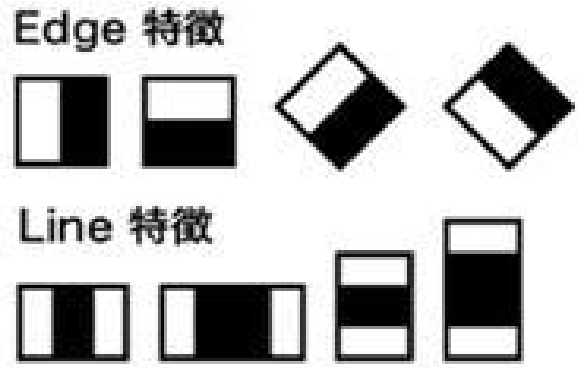

図 1 Haal-Like 特徴の例

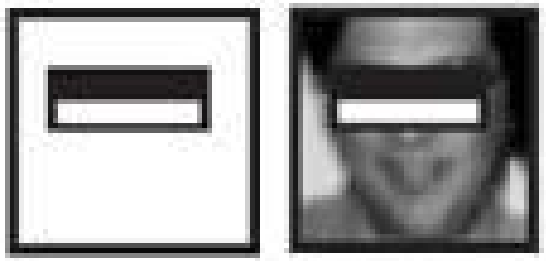

図 2 haar-like 特徴と画像を重ね合わせた例

3. GPU での高速化

本章では，GPU での高速化をするにあたっての制限を述べる．3.1 節では GPU の構造と処理単位について，3.2 節では GPU での制限について述べる．

3.1 GPU の構造と処理単位

使用する GPU である Tesla C1060 には，SM (Streaming Multiprocessor) が 30 個，SP (Streaming Processor) が 240 個装備されている．Tesla C 1060 では，NVIDIA 社が無償で提供している汎用処理向け統合開発環境である CUDA により，C 言語でプログラム開発を行うことができる．CUDA の処理単位として，グリッド，ブロック，スレッドの 3 種類がある．グリッドとは CPU から GPU へ渡す処理単位のことで，ブロックとは GPU 内の各 SM で処理される単位，スレッドとは SM 内で処理される単位のことである．

図 4 のようなグリッドが複数あり，そのそれぞれのグリッド内に複数のブロックが存在し，さらに各ブロックの中に多数のスレッドが生成され，並列処理が行われる．ブロックは，1 つのグリッドに最大 65535 個，スレッドは，Tesla C1060 では最大 512 個である．また，CPU から GPU へ処理を渡す際には，スレッド数とブロック数を指定する必要がある．

3.2 GPU での制限

CPU と GPU はそれぞれ独立したメモリを持っている．CUDA においては，CPU から

Vol.2011-MPS-82 No.21
2011/3/8

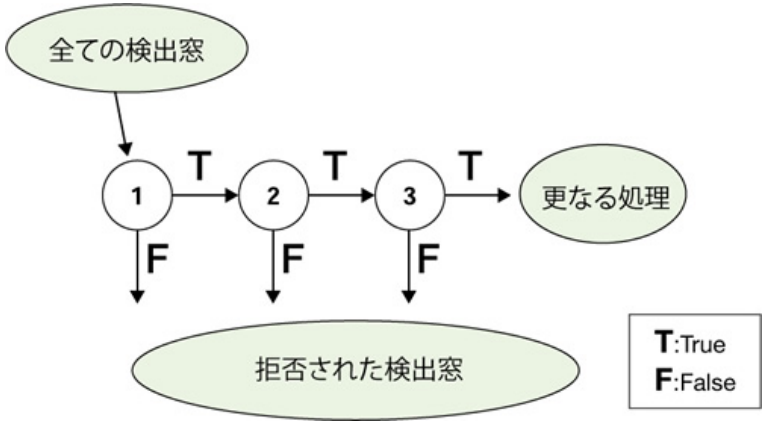

図 3 カスケード

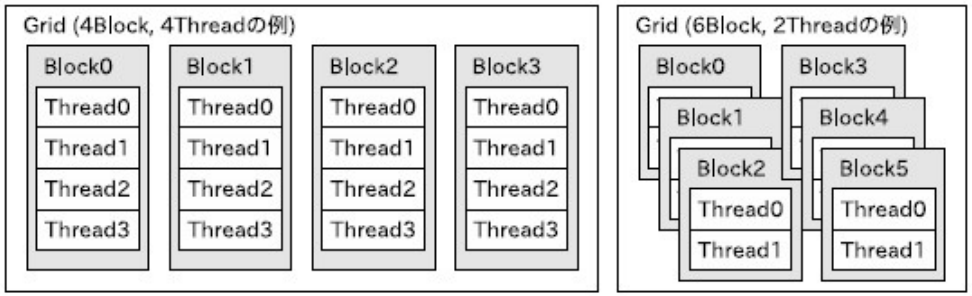

図 4 グリッド・ブロック・スレッドの概念図

GPU 上のメモリに対する読み書きには制限があり，GPU から CPU 上のメモリを直接操作することはできない．CPU から GPU へのデータの読み書き，すなわち，CPU-GPU 間のデータ転送については，カーネル関数の前後で CUDA の API を用いて行う必要がある．

GPU におけるメモリの分類を図 5 に示す．GPU には複数の種類のメモリが搭載されており，それぞれが異なる特性を持っている．各メモリを適切に利用することが GPU を用いて高い性能を得る上で非常に重要である．ここで，特に使用する機会が多いと思われるグローバル・メモリとシェアード・メモリについて述べる．グローバル・メモリは，GPU 上の全ての SP で共有するメモリであり，容量は 4GB と多く，全ブロック，全スレッドから

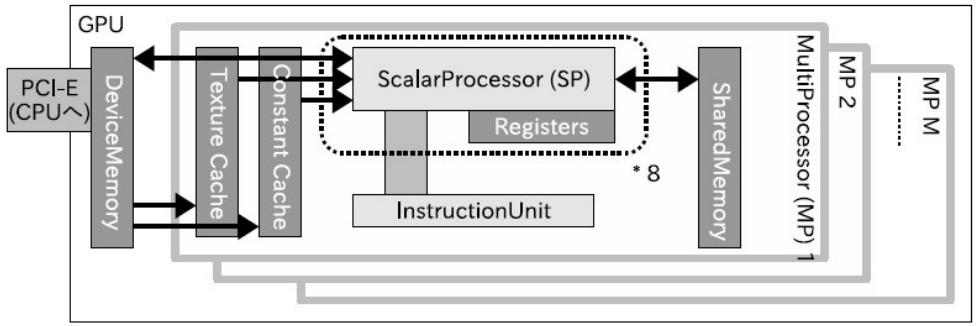

図 5 GPU のメモリの概念図

共有メモリとして利用可能なメモリである．また，連続アクセスに対しては高速だが，ランダムアクセスに対しては低速であり，シェアード・メモリに比べると約 100 倍以上もアクセスが低速である．一方，シェアード・メモリは，各 SM ごとに搭載されており，容量は 16KB と少なく，CPU から直接値を読み書きすることはできない．また，ブロック内全てのスレッドから共有でき，アクセスが高速である．メモリが 16 個のバンクで構成されており，同時に複数のスレッドが同一のバンクにアクセスすると性能が劣化する (バンクコンフリクト)．

CUDA 対応 GPU では，ある命令を実行する際には，32 個のスレッドがワープという単位で管理されている．したがって，ワープ内の各スレッドが「if 文」などの制御構文で異なる方向に分岐することを繰り返す場合，GPU は多くのルートを実行することになる．これは，ワープ・ダイバージェントと呼ばれ，GPU の性能低下を招く．

したがって，これらの特徴を考慮してデータや処理命令を扱うことが GPU から高い性能を引き出すポイントとなる．

4. 実装と評価

本章では，CPU と GPU での実装方法の違いを説明し，実装結果と評価について述べる．4.1 節では CPU 実装の場合の処理の流れについて述べ，4.2 節では GPU での処理について，4.3 節では実装結果について述べ，4.4 節でその評価を述べる．

4.1 CPU 実装の場合の処理の流れ

CPU 実装の場合の処理の流れは，以下の通りである．

(1) 検出対象画像を読み込む
(2) 画像ピラミッドを作成する
(3) 乱数列を発生させる
(4) 画像ピラミッドの下段から順に走査開始
(5) 検出された顔の位置を全て矩形で囲む
(6) 矩形で囲んだ画像を表示

まず，検出したい画像を読み込む（1）．次に，様々な大きさの顔を検出するために，読み込んだ画像の画像ピラミッドを作成する（2）．ここでは，1.25 倍ずつ 4 段階まで縮小した．次に，乱数列を発生させる（3）．これは，GPU で並列処理を行いやすくするために利用するもので，ピラミッド画像上を検索窓が走査する際に，この乱数列から決定した位置を走査開始点にするためのものである．このときの走査の概念図を図 6 に示す．その後，画像ピラミッドの下段から順に，(3) から決定した走査開始点から検索窓を走査させる（4）．その際に，検索窓ごとに Haar-Like 特徴を計算していく．このとき，走査範囲は 100 × 100 pixel とし，走査開始点は 100 回変更して走査を行う．走査の結果，顔であると判断された位置を矩形で囲む（5）．その後，その画像を表示する（6）．画像の読み込み，矩形の描画，画像の表示の処理には，OpenCV1.2 を使用する．

4.2 GPU での処理

GPU には，4.1 節の（4）の処理を実行させる．この理由として，(4) での処理が CPU 実装の際に最も処理に時間がかかっていること，異なるデータに対して同じ演算を繰り返しているため，並列処理に適していることが挙げられる．

処理（4）において，走査範囲の画像上での処理全体を 1 つのブロックに割り当てる．そして，各ブロック内のスレッド数は最大の 512 として実装する．このとき，全ブロック数は 4.1 節での走査開始点の変更回数と対応するので 100 となる．また，データの格納場所について考える．処理（4）で使用する配列データには，以下のものがある．

(1) ピラミッド画像中の 1 つの画像データ配列

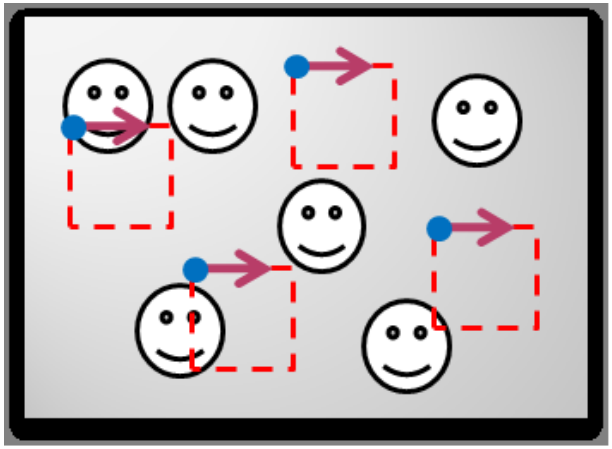

図 6 走査の概念図

(2) 走査範囲内の画像データ配列
(3) 走査済か否かのフラグ配列
(4) Haar-Like 画像データ配列
(5) 乱数配列

このうち，他のブロックとデータを共有しない，かつ，参照回数の多いデータは（b）である．したがって，(b) のデータはシェアード・メモリに転送する．その他のデータは，全ブロックで共通して参照するので，グローバル・メモリに格納する．また，GPU 上の配列は全て 1 次元配列として，連続アクセスができるようにする．

4.3 実装結果と評価

実装には，CPU は Intel core i7 を，GPU は Tesla C1060 を使用する．CPU での実装結果を表 1 ，CPU での実装結果を表 2 に示す．表中の番号は，4.1 節の処理番号と対応している．

プログラム全体の処理時間に注目すると，約 980.51 倍高速化されている．また，処理 (4) を GPU で実行させたカーネル関数の部分に注目すると，2730.31 倍高速化されている．CPU での（ 4 ）の処理は，走査開始点から指定範囲を走査して Haar-Like 特徴を計算して

Vol.2011-MPS-82 No.21
2011/3/8

図 7 実装に用いた画像

いく処理を 200 回ループさせる必要があるが，GPU で並列処理を行うことで，この処理が同時に行える．さらに，指定範囲を走査する処理の際にも，各スレッドが同時に Haar-Like 特徴を計算できるため，より高速な処理が可能となったと考えられる．また，カーネル関数の処理時間に対して，データ転送時間が大きいことが分かる．

5. ま と め

本稿では，複数人物の顔を検出するプログラムを，GPGPU 用ハードウェア Tesla C1060 を用いた高速化手法の一提案を行った．顔検出手法には，haar-like 特徴を用いた AdaBoost ベースの識別器を利用した．

高速化手法として，読み込んだ画像に対して，ランダムに走査開始点を決め，そこから指定した範囲内を検索窓が走査しながら Haar-Like 特徴を計算していく処理を行う際に，指

表 1 CPU での実行時間

処理 (1)〜(6)	処理 (4)
532223.14[ms]	108093.08[ms]

表 2 GPU での実行時間

処理 (1)〜(6)	カーネル関数	データ転送
542.80[ms]	39.59[ms]	64.28[ms]

定範囲の画像領域での処理をブロックに割り当て，ブロック内の 1 つのスレッドが 1 画素に対する処理を行うようにすることで高速化を図った．実装の結果，カーネル関数部で約 2730.31 倍，全体としては約 980.51 倍の高速化ができた．

今後は，バンクコンフリクトやワープの観点からの高速化や，顔検出自体の精度を向上させる必要がある．また，顔検出の際には，カスケードや Integral Image と呼ばれる画像の合計値を高速に求める計算法があるが，これらを導入することでもより処理時間が短縮されると考えられる．

参 考 文 献

1) Yoav Freund and Robert E. Schapire ”A decision-theoretic generalization of on-line learning and an application to boosting ” In Computational Learning Theory: Eurocolt ’95, pages 23-27. Springer-Verlag, (1995)
2) Paul Viola and Michael J. Jones, ”Rapid Object Detection using a Boosted Cascade of Simple Features ”, IEEE CVPR, (2001)
3) Yoav Freund and Robert E. Schapire ”A decision-theoretic generalization of on-line learning and an application to boosting ” In Computational Learning Theory: Eurocolt ’95, pages 23-27. Springer-Verlag, (1995)

Vol.2011-MPS-82 No.21
2011/3/8

4) C. Papageorgiou, M. Oren, and T. Poggio. ”A general framework for object detection. ”In International Conference on Computer Vision, (1998)