Vol.2013-HPC-138 No.18
2013/2/22

医用画像位置合わせを対象にした結合ヒストグラム生成のGPUによる高速化

池田 圭[1]　伊野 文彦[1]　萩原 兼一[1]

概要：本稿では，医用画像位置合わせのための結合ヒストグラム生成を GPU（Graphics Processing Unit）により高速化する手法を提案する．提案手法は，結合ヒストグラムのデータサイズを削減し共有メモリ上に格納する．さらに，結合ヒストグラムのビンがオーバフローすることを防ぐために，共有メモリ上のデータをグローバルメモリに出力するフラッシュ機構を作成し，データを出力する際のグローバルメモリアクセス量を削減した．実験では，$512 \times 512 \times 296$ ボクセルからなる臨床画像を用いて提案手法の性能を評価した．結果として，Fermi アーキテクチャの GPU において，提案手法によるカーネルは，グローバルメモリのみを用いたカーネルと比較して約 13 倍の速度向上を得た．一方，Kepler アーキテクチャの GPU においては，1.3 倍〜2 倍の速度向上を得た．Fermi アーキテクチャの GPU において，提案手法による画像位置合わせの実装は，グローバルメモリのみを用いて結合ヒストグラムを生成する実装と比較して，約 8 倍の速度向上を得た．

キーワード：結合ヒストグラム，医用画像位置合わせ，GPU，CUDA

Accelerating Joint Histogram Computation for Medical Image Registration using the GPU

Kei Ikeda[1]　Fumihiko Ino[1]　Kenichi Hagihara[1]

Abstract: This paper presents an acceleration method for joint histogram computation used in medical image registration using a graphics processing unit (GPU). Our method reduces the data size of joint histograms so that they can be stored in global memory. Furthermore, we implement a flush mechanism to avoid overflows of bins on joint histograms. This mechanism outputs the data on shared memory to global memory. Our method also reduces the amount of global memory access. In experiments, we evaluate our method with clinical datasets having $512 \times 512 \times 296$ voxels. As a result, our kernel is approximately 13 times faster than a kernel using only global memory on the Fermi architecture GPU. On the other hand, our kernel achieves speedup of 1.3 to 2 on the Kepler architecture GPU. Our implementation of image registration is approximately 8 times faster than an implementation using only global memory to generate joint histograms.

Keywords: joint histogram, medical image registration, GPU, CUDA

1. はじめに

医用画像位置合わせ [1] とは，患者の体位や呼吸により生じる画像の位置ずれを補正する技術である．この技術は，画像を用いた診断および手術支援において重要な役割を果たす．例えば，撮影時刻の異なる 2 系列の画像を位置合わせし，これらの濃度差分を計算することにより，腫瘍の状態変化を検出できる [2]．さらに，CT（Computer Tomography），MR（Magnetic Resonance），および PET（Positoron Emission Tomography）などの異なるモダリティ間の画像を位置合わせし，これらの画像を重ね合わせることにより，患部の詳細な情報を得ることができる [3]．

[1] 大阪大学大学院情報科学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Information Science and Technology, Osaka University

多くの位置合わせアルゴリズムは画像間の類似度を定義し，類似度をコスト関数とする最適化問題に帰着している．画像のモダリティに依存しないコスト関数として，情報理論に基づく相互情報量および正規化相互情報量 [4] が広く用いられている．これらの計算は，画像間の結合ヒストグラムを必要とする．しかし，結合ヒストグラムの生成はメモリ参照量の多い処理であるために，高速化が必要である．

そこで，GPU（Graphics Processing Unit）向け統合開発環境 CUDA（Compute Unified Device Architecture）[5] を用いて，結合ヒストグラム生成を並列処理することにより高速化する研究が行われている [6], [7], [8], [9].

医用画像から生成した結合ヒストグラムのサイズは一般に 256 K バイト以上であり，高々 48 K バイトの共有メモリに結合ヒストグラムの全体を格納できない．したがって，既存の高速化手法 [6], [7], [8], [9] は，グローバルメモリ上に結合ヒストグラムを格納してきた．

本研究は共有メモリを用いることにより，画像位置合わせのための結合ヒストグラムを高速に生成する手法を提案する．提案手法は，共有メモリ上に結合ヒストグラムを格納できるようにデータ構造を工夫し，結合ヒストグラムのサイズを削減する．さらに，結合ヒストグラムのビンにおけるオーバフローを防ぐために，共有メモリ上のデータを適宜グローバルメモリに出力するフラッシュ機構を作成し，データを出力する際のグローバルメモリアクセス量を削減する．

以降では，まず 2 節で相互情報量に基づく医用画像位置合わせについてまとめ，3 節で解決すべき問題点を示す．次に 4 節で提案する高速化手法を示し，5 節で評価実験の結果を示す．最後に 6 節で今後の課題とともに本稿をまとめる．

2. 相互情報量に基づく医用画像位置合わせ

位置合わせ対象とする参照画像を R とし，浮動画像を F とする．また，画像 R および F 間の類似度を $S(R,F)$ とする．このとき，画像位置合わせは $S(R,T(F))$ を最大化する幾何変換 $T:(x,y,z)\mapsto(x',y',z')$ を探索する問題とみなせる．ここで，T は座標 (x,y,z) 上のボクセルを座標 (x',y',z') に変換するものである．一般に，探索には反復法を用いるため，位置合わせの過程で $S(R,T(F))$ を繰り返し計算する．

モダリティの異なる画像位置合わせを実現するために，類似度関数 S として相互情報量および正規化相互情報量が広く用いられる．画像 R および F の相互情報量 $S_{MI}(R,F)$ および正規化相互情報量 $S_{NMI}(R,F)$ は，式 (1) および式 (2) で与えられる．

$$S_{MI}(R,F)=H(R)+H(F)-H(R,F) \tag{1}$$

$$S_{NMI}(R,F)=\frac{H(R)+H(F)}{H(R,F)} \tag{2}$$

ここで，$H(R)$ は R のエントロピーであり，$H(R,F)$ は R および F の結 合エントロピーである．各々は式 (3) および式 (4) で与えられる．

$$H(R)=-\sum_{r\in R}p_R(r)\log p_R(r) \tag{3}$$

$$H(R,F)=-\sum_{r\in R,f\in F}p_{RF}(r,f)\log p_{RF}(r,f) \tag{4}$$

ここで，r および f はそれぞれ画像 R および F の輝度値である．画像 R を輝度値 r の離散確率変数とみなせば，$p_R(r)$ は R の周辺確率分布であり，$p_{RF}(r,f)$ は R および F の同時確率分布である．$p_R(r)$ の算出には，輝度値 $r\in R$ のヒストグラムが必要である．同様に，同時確率分布 $p_{RF}(r,f)$ の算出には，輝度値の組 $\langle r,f\rangle$（$r\in R$ および $f\in F$）に関する結合ヒストグラムが必要である．

画像 R において座標 (x,y,z) に位置するボクセルの輝度値を $R(x,y,z)$ とおく．このとき，結合ヒストグラムは $\langle R(x,y,z),F(x,y,z)\rangle$ を ビンとするヒストグラムである．

d 階調の画像が与えられたとき，結合ヒストグラムのビン数は d^2 となる．したがって，ビンのサイズが b であるとき，結合ヒストグラムのサイズは d^2b となる．

3. 解決すべき問題点

一般に，医用画像では $d\geq 256$ であるため，$b=4$ バイトであるとき，結合ヒストグラムのサイズは少なくとも 256 K バイトになる．そのために，高々 48 K バイトの共有メモリに結合ヒストグラムの全体を格納することはできない．したがって，既存の高速化手法 [6], [7], [8], [9] はグローバルメモリ上に結合ヒストグラムを格納している．

グローバルメモリ上に結合ヒストグラムを格納した場合は，結合ヒストグラムを更新するためのメモリアクセスが性能低下の原因になる可能性がある．性能低下の原因になり得るメモリアクセスパターンは以下の 2 通りである．

(A)　アトミック演算に起因する逐次書きこみ

(B)　グローバルメモリへのランダムアクセス

パターン（A）は，同一のメモリ番地に複数のスレッドが同時に書き込む可能性があることに起因する．この場合，アトミック演算を用いて排他的にヒストグラムを更新する必要がある．結果として，一連の書き込みは逐次処理されてしまい，並列処理の効率が低下する．

そこで，Shams ら [6] はスレッドごとに独立な結合ヒストグラム（局所ヒストグラム）を保持し，書きこみ先の重複を除去している．これによりアトミック演算を使うことなく輝度値を計数できる．ただし，最終的な結合ヒストグラムを得るためには，複数の局所ヒストグラムを 1 つに集約する必要がある．また，パターン（B）による性能低下は未解決である．さらに，彼らの手法はスレッドごとに局

所ヒストグラムを保持しているため，グローバルメモリ使用量が多い．

Chen [7]，Shams ら [8]，および Vetter ら [9] は，前処理として画像のボクセルを輝度値順にソートすることにより，パターン（B）による性能低下を回避している．しかし，ソートによるオーバヘッドが大きく，ソートのために画像と同じサイズのグローバルメモリ領域を必要とする．

4. 提案手法

提案手法は以下の 2 つの工夫により結合ヒストグラムの生成を高速化する．

(I) 結合ヒストグラムのデータサイズを削減し，結合ヒストグラムを共有メモリ上に格納する．

(II) 結合ヒストグラムのビンにおけるオーバフローを防ぐために，共有メモリ上のデータを適宜グローバルメモリに出力するフラッシュ機構を作成し，データを出力する際のグローバルメモリアクセス量を削減する．

以下，（I）および（II）について詳細に説明する．

4.1 結合ヒストグラムのデータサイズ削減

図 1 に，実際の医用画像を用いて得た結合ヒストグラムを示す．図が示すように，結合ヒストグラムは疎なデータである．さらに，画像間の類似度が高くなるにつれて（位置合わせが進むにつれて），結合ヒストグラム上の計数値は対角線上に収束する．したがって，対角線を中心とする部分領域のみを格納すれば，データサイズを削減できる．また，収束とともに部分領域は小さくなる可能性が高い．

図 2(a) に示すように，長さ W の底辺を持ち中心が対角線を貫く高さ $d-1$ の平行四辺形 $P(W)$ を考える．すべての計数値が $P(W)$ の領域内に存在すると仮定する．このとき，提案手法は $P(W)$ 内部のビンのみを図 2(b) に示すデータ構造に格納する．そのために，単純なデータ構造における位置 (r, f) のビンを次式に基づいて並び替える．

$$(r', f') = (r, f - r + W/2) \tag{5}$$

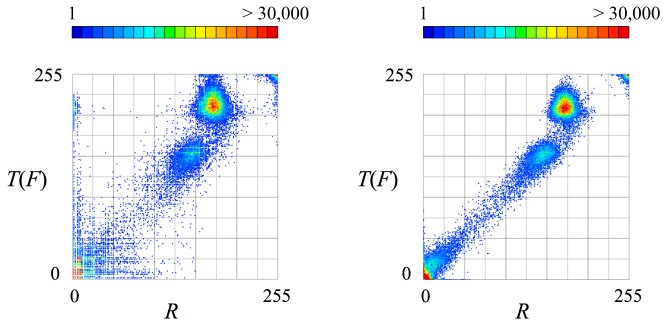
ここで，(r', f') は提案するデータ構造における位置 (r', f') のビンを表す．

位置 (r, f) に存在するビンが $P(W)$ の領域内に存在するか否かを判定するためには，次式の真偽を調べればよい．

$$r - W/2 \le f \le r + W/2 \tag{6}$$

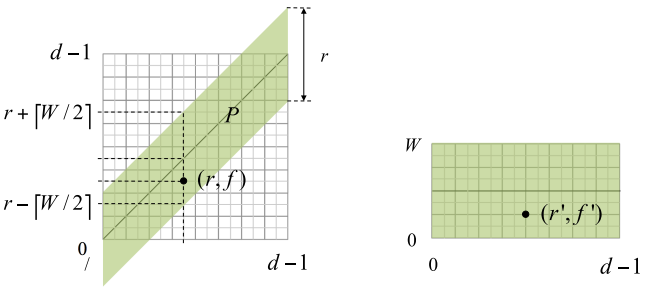
提案手法は式 (6) の真偽に応じて 2 つの手法を使い分ける．

- 真のとき，各スレッドブロックは共有メモリ上に平行四辺形 $P(W)$ を局所ヒストグラムとして持ち，自身が担当する $P(W)$ 内のビンのみを更新する．



(a) 位置合わせ前　(b) 位置合わせ後

図 1 実際の医用画像から得た結合ヒストグラム



(a) 単純なデータ構造　(b) 提案するデータ構造

図 2 結合ヒストグラムのデータ構造

- 偽のとき，各スレッドブロックはグローバルメモリ上に局所ヒストグラムを持ち，自身が担当する局所ヒストグラム内のビンを更新する．

各スレッドブロックは，画像 R および F をブロック分割した部分画像 $\mathcal{R}_{i,j,k}$ および $\mathcal{F}_{i,j,k}$ から，局所ヒストグラムを生成する．スレッドブロック内の各スレッドは $\mathcal{R}_{i,j,k}$ および $\mathcal{F}_{i,j,k}$ から輝度値を 1 つ読み込み，自身が属するスレッドブロックが持つ局所ヒストグラムを更新する．最後に，各スレッドブロックが持つ局所ヒストグラムを 1 つに集約することで最終的な結合ヒストグラムを得る．最終的な結合ヒストグラムはグローバルメモリ上に格納する．なお，集約は 2 分木状に並列処理できる．

W は画像の階調 d，ビンのサイズ b，および共有メモリの容量に依存して決まる．$d = 256$ かつ $b = 1$ バイト（ビンのデータ型が unsigned char 型）である場合，$P(W)$ のサイズは $256W$ バイトとなる．したがって，共有メモリの容量が 48 K バイトである場合，$W < 192$ である必要がある．

4.2 フラッシュ機構

$P(W)$ のサイズを $W < 192$ を満たす範囲で最大化するためには，b を最小化する必要がある．しかし，b を小さくすることにより，結合ヒストグラムの生成時にビンがオーバフローする可能性がある．オーバフローを防ぐためには，共有メモリ上の局所ヒストグラムを，グローバルメモ

```
generateJointHistogram (R, F, X, Y, Z, h)

    R, F: 参照画像および浮動画像
    X, Y, Z: R および F のサイズ
    h: グローバルメモリ上にある結合ヒストグラム用の
       メモリ領域

 1: スレッドID およびスレッドブロックID に基づいて
    スレッドの担当領域を決定
 2: 共有メモリ上に結合ヒストグラム用のメモリ領域 hs
    を確保
 3: hs を初期化
 4: スレッドブロック内バリア同期

 5: for z ← 0 to Z − 1 do
 6:     r ← R(x, y, z)
 7:     f ← F(x, y, z)
 8:     f' ← f – r + ⌊W/2⌋
 9:     if hs(r, f') = 0xff then
10:         val ← atomicExch(hs(r, f'), 0)
11:         atomicAdd(h(r, f), val)
12:     end if
13:     atomicAdd(hs(r, f'), 1)
14:     スレッドブロック内バリア同期
15: end for

16: atomicAdd 命令を用いて hs を h に集約
```

図 3 提案するフラッシュ機構を用いた結合ヒストグラム生成の疑似コード

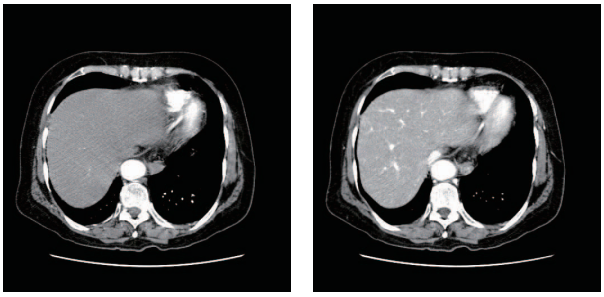
リ上の結合ヒストグラムに適宜書き出すフラッシュ機構が必要である．

提案機構は，局所ヒストグラムを更新する際にビンのオーバフローを検出し，オーバフローするビンのみをグローバルメモリ上の結合ヒストグラムに出力する．一部のビンのみを出力することにより，局所ヒストグラム全体を出力する場合と比較して，グローバルメモリへのアクセス量を削減することができる．このフラッシュ機構を用いた結合ヒストグラム生成の疑似コードを図 3 に示す．

図 3 の atomicExch 命令（10 行目）は，複数のスレッドが同一ビンのオーバフローを同時に検知した場合に必要である．なお，CUDA のアトミック演算は，32 ビット変数および 64 ビット変数にのみ対応している [5]．そのため，ビンのサイズが 32 ビット未満である場合は，アトミック演算と同機能のデバイス関数を独自に実装する必要がある．図 4 に，8 ビット変数に対して atomicAdd 命令と同等の処理を行うデバイス関数の実装例を示す．この関数は，メモリの 32 ビット境界に対する 8 ビット変数の距離を求め，距離に応じて書きこむ数値を左にシフトする．その後，32 ビット境界にアライメントされたメモリ領域に対して，シフトした数値を atomicAdd 命令により書きこむ．たとえば，書きこむ数値が $11_{(16)}$ であり，変数が境界に対して 16 ビット離れている場合，この数値を 16 ビットだけ左にシフトした数値（$00110000_{(16)}$）をメモリ領域に書きこむ．

```
 1: __device__ void uchar_atomicAdd (uchar*  address,
 2:                                  uchar  val)
 3: {
 4:     size_t  remainder = (size_t)address&3;
 5:     uint  *uint_address = (uint *)((size_t)address –
 6:                                    remainder);
 7:     uint  uint_val = (uint)val << (remainder * 8);
 8:
 9:     atomicAdd(uint_address, uint_val);
10: }
```

図 4 atomicAdd 命令と同等の処理を 8 ビット変数に対して行うデバイス関数の実装例



(a) 参照画像 R　　(b) 浮動画像 F

図 5 CT 画像のスライスの例

5. 評価実験

提案手法による速度向上を実験により評価する．実験では，撮像時刻の異なる 2 系列の腹部 X 線 CT 像を用いて，結合ヒストグラム生成および医用画像位置合わせの実行時間を計測した．

画像のサイズは $512 \times 512 \times 296$ ボクセルであり，ボクセル間隔は $0.67 \times 0.67 \times 0.67$ ミリである．各ボクセルは 8 ビットの値を持つ（$d = 256$）．画像のスライスの例を図 5 に示す．

実験に用いた計算機は，CPU として Intel Core i7 3930K 3.2 GHz を持つ．主記憶容量は 16 G バイトであり，OS は Windows 7 である．GPU として NVIDA GeForce GTX 580，GeForce GTX 680 および Tesla K20 を使用する．GeForce GTX 580 は Fermi アーキテクチャの GPU であり，GeForce GTX 680 および Tesla K20 は Kepler アーキテクチャの GPU である．なお，CUDA のバージョンは 5.0 である．

5.1 結合ヒストグラム生成の評価実験

まず，提案手法による結合ヒストグラム生成の速度向上を評価する．そのために，以下の 3 種類のカーネルを用意し，それらの実行時間を測定した．3 種類のカーネルはいずれもスレッドブロックごとに局所ヒストグラムを保持する．

（1） 局所ヒストグラムをグローバルメモリ上に持つ．

表 1 各カーネルの実効メモリ帯域幅（単位：G バイト/s）

GPU	ピーク性能	実効性能		
		カーネル（1）	カーネル（2）	カーネル（3）
GeForce GTX580	192.4	6.7	10.4	83.9
GeForce GTX680	192.2	58.6	9.2	74.5
Teska K20	208.0	42.5	10.6	84.6

（2） 局所ヒストグラムを共有メモリ上に持つ（4.1 節）．ビンのオーバフローを検出した場合，局所ヒストグラム全体をグローバルメモリに書き出す．

（3） 局所ヒストグラムを共有メモリ上に持つ（4.1 節）．ビンのオーバフローを検出した場合，オーバフローするビンのみをグローバルメモリに書き出す（4.2 節）．

CT 像は 62 例あり，これらから生成した結合ヒストグラムの計数値は，すべて $W = 176$ の平行四辺形 $P(W)$ 内に存在する．

図 6 に，実行時間の比較を示す．図中の実行時間は 62 例の平均値である．なお，実行時間はすべて平均値 ± 2 ミリ秒の範囲に収まっている．

GeForce GTX 580 を用いた場合，カーネル（1）の実行時間は 139 ミリ秒であり，カーネル（2）の実行時間は 89.4 ミリ秒である．したがって，結合ヒストグラムを共有メモリ上に格納することにより 1.6 倍の速度向上を得ている．さらに，カーネル（3）の実行時間は 11.1 ミリ秒であり，カーネル（2）と比較すると，8 倍の速度向上を得ている．この速度向上は提案したフラッシュ機構によるものである．結果として，提案手法によるカーネル（3）はグローバルメモリのみを用いたカーネル（1）と比較して 13 倍の速度向上を得た．

GeForce GTX 680 を用いた場合，カーネル（1）の実行時間は 15.9 ミリ秒であり，カーネル（2）の実行時間は 101 ミリ秒であった．したがって，カーネル（2）はカーネル（1）と比較して 6.4 分の 1 に速度低下している．さらに，カーネル（3）の実行時間は 12.5 ミリ秒であり，カーネル（1）と比較して 1.3 倍の速度向上にとどまった．同じアーキテクチャの GPU である Tesla K20 を用いた場合も，これらと同様の結果が得られた．

Fermi アーキテクチャの GPU を用いた場合と比較して，Kepler アーキテクチャの GPU を用いた場合のカーネル（1）が高速である理由は，アーキテクチャ間でグローバルメモリに対するアトミック演算の速度が異なるためである．Kepler アーキテクチャの GPU では，L2 キャッシュの性能向上によりグローバルメモリに対するアトミック演算の速度が向上した [10]．GeForce GTX 680 を用いた場合，グローバルメモリに対するアトミック演算の速度は，GeForce GTX 580 と比較して最大で 9 倍高速である．一方で，共有メモリに対するアトミック演算の性能には明確な差がない．

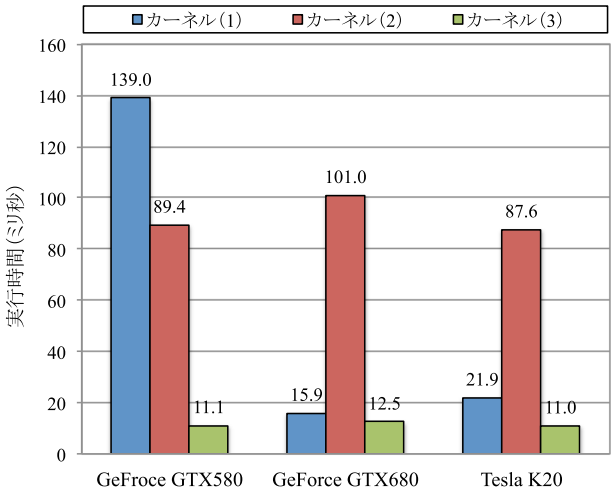


図 6 結合ヒストグラム生成の実行時間の比較

次に，ピーク性能に対する各カーネルの実効性能を調べた．結合ヒストグラム生成では，メモリアクセスが実行時間の大半を占めるため，性能の指標としてメモリ帯域幅を用いた．表 1 に結果を示す．結果から，提案手法による実効メモリ帯域幅はピーク性能の 39〜44%であった． 4 節で述べたように，位置合わせが進むにつれて結合ヒストグラム上の計数値は対角線上に収束する．これは，多数のスレッドがアトミック演算を用いて同一のビンを更新することを意味する．したがって，計数値の分布に依存しないメモリアクセスパターンによりビンを更新できれば，性能を改善できる見込みがある．

5.2 医用画像位置合わせの評価実験

最後に，提案手法による医用画像位置合わせの速度向上を実験により評価する．画像位置合わせのアルゴリズムとして，Rueckert ら [2] のアルゴリズムを用いる．彼らのアルゴリズムは類似度関数 S として正規化相互情報量を用いる．また，幾何変換 T として B-spline 関数 [11] に基づく変換を用いる．この変換は，画像領域に制御点を網の目状に配置し，制御点の変位により物体を変形する．最適化には最急降下法を用いる．さらに，計算量を削減するために，画像および制御網の解像度を階層化する．

実験では，4 例の CT 像を用いて，肝臓の位置合わせを試みた．その際，以下の 2 種類の実装を用意し，それらの実行時間を計測した．2 種類の実装は，結合ヒストグラム

表 2　各階層のパラメータ

階層レベル	1	2	3
ボクセルのサイズ (mm)	2.68	1.34	0.67
ボリュームサイズ (ボクセル)	128 × 128 × 74	256 × 256 × 148	512 × 512 × 296
δ：制御点間隔 (mm)	42.88	21.44	10.72

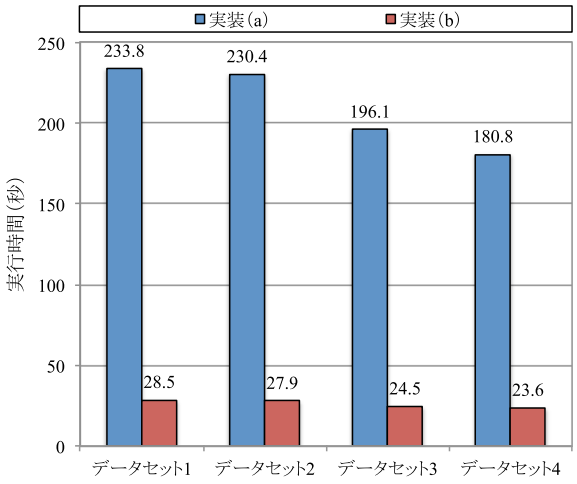


図 7　医用画像位置合わせの実行時間の比較

生成の手法のみが異なる．

(a)　カーネル（1）と同様に結合ヒストグラムを生成する．

(b)　カーネル（3）と同様に結合ヒストグラムを生成する．

画像および制御網の解像度は 3 層構造とした（表 2）．なお，GPU として GeForce GTX 580 のみを用いた．

図 7 に，実行時間の比較を示す．実装（b）の実行時間は 23.6〜28.5 秒であり，実装（a）の実行時間 180.8〜233.8 秒と比較して約 8 倍の速度向上を得ている．このように，いずれのデータセットに対しても，提案手法は画像位置合わせを高速化できている．

6. まとめと今後の課題

本稿では，医用画像位置合わせのための結合ヒストグラムを GPU により高速に生成する手法を提案した．提案手法は，結合ヒストグラムのサイズを削減し，共有メモリ上に格納した．さらに，結合ヒストグラムのビンにおけるオーバフローを検出し，オーバフローするビンのみをグローバルメモリに出力するフラッシュ機構により，グローバルメモリへのアクセス量を削減した．

実験では，512 × 512 × 296 ボクセルからなる臨床画像を用いて提案手法の性能を評価した．結果として，グローバルメモリのみを用いたカーネルと比較して，Fermi アーキテクチャの GPU において約 13 倍の速度向上を得た．一方，Kepler アーキテクチャの GPU においては 1.3 倍〜2 倍の速度向上にとどまった．また，Fermi アーキテクチャの GPU において，提案手法による画像位置合わせの実装は，グローバルメモリのみを用いて結合ヒストグラムを生成する実装と比較して，約 8 倍の速度向上を得た．

今後の課題は，輝度値の分布に依存しない結合ヒストグラムの生成により，さらなる高速化を図ることである．

謝辞　本研究の一部は，JST CREST「進化的アプローチによる超並列複合システム向け開発環境の創出」，科研費 23300007，および 23700057 の補助による．

参考文献

[1] J. V. Hajnal, D. L. Hill, and D. J. Hawkes, Eds., Medical Image Registration. Boca Raton, FL: CRC Press, 2001.
[2] D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. Leach, D. J. Hawkes, “ Nonrigid registration using free-form deformations: Application to breast MR images, ” IEEE Trans. Medical Imaging, vol. 18, no. 8, pp. 712—721, Aug. 1999.
[3] F. Maes, A. Collignon, D. Vandermeulen, G. Marchal, P. Suetens, “ Multimodality image registration by maximization of mutual information, ” IEEE Trans. Medical Imaging, vol. 16, no. 2, pp. 187—198, Apr. 1997.
[4] C. Studholme, R. T. Constable, and J. S. Duncan, “ Accurate alignment of functional EPI data to anatomical MRI using a physics-based distortion model, ” IEEE Trans. Medical Imaging, vol. 19, no. 11, pp. 1115—1127, Nov. 2000.
[5] NVIDIA Corporation, “ CUDA Programming Guide Version 5.0, ” Oct. 2012. https://developer.nvidia.com/
[6] R. Shams and R. A. Kennedy, “ Efficient histogram algorithms for NVIDIA CUDA compatible devices, ” in Proc. Int ’ l Conf. Signal Processing and Communications Systems (ICSPCS ’ 07), Dec. 2007, pp. 418—422.
[7] S. Chen, J. Qin, Y. Xie, W.-M. Pang, and P.-A. Heng, “ CUDA-based acceleration and algorithm refinement for volume image registration, ”in Proc. 8th IEEE Int ’l Conf. Future BioMedical Information Engineering (FBIE ’ 09), Dec. 2009, pp. 544—547.
[8] R. Shams, P. Sadeghi, R. A. Kennedy, and R. I. Hartley, “ A survey of medical image registration on multicore and the GPU, ” IEEE Signal Processing Magazine, vol. 27, no. 2, pp. 50—60, Mar. 2010.
[9] C. Vetter and R. Westermann, “ Optimized GPU histograms for multi-modal registration, ” in Proc. 8th IEEE Int ’ l Symp. Biomedical Imaging (ISBI ’ 11), Apr. 2011, pp. 1227—1230.
[10] NVIDIA Corporation, “ GTX 680 Kepler Whitepaper, ” Mar. 2012. https://developer.nvidia.com/
[11] S. Lee, G. Wolberg, and S.Y. Shin, “Scattered data interpolation with multilevel B-splines,” IEEE Trans. Visualization and Computer Graphics, vol.3, no.3, pp.228–244, July 1997.