Vol.2012-DPS-150 No.24
Vol.2012-CSEC-56 No.24
2012/2/29

kd-tree により構造化された 大規模高次元ベクトル群に対する GPU を用いた高速最近傍探索法の検討

松村聖司[†]　毛受崇[†]　赤間浩樹[†]　松尾嘉典[†]
奥村昌和[†]　山室雅司[†]

局所特徴量を用いた物体認識処理での高次元ベクトル群の最近傍探索処理において，ベクトル間のユークリッド距離計算が行われている．データベース中のベクトル数が大規模になると，最近傍探索の処理時間は膨大になる．本検討では，kd-tree によるデータ構造化と GPU でのユークリッド距離計算の並列化を用いることで，探索精度の低下を抑えた最近傍探索処理高速化の手法を提案する．

Nearest Neighbor Search using GPU for Massive and High-dimensional Vectors Structured by kd-tree

Seiji Matsumura[†], Takashi Menjo[†], Hiroki Akama[†],
Yoshinori Matsuo[†], Masakazu Okumura[†],
and Masashi Yamamuro[†]

When the total number of local characteristic high-dimensional vectors in object recognition becomes massive, it takes long time to process nearest neighbor search. In this study, we propose a method that makes nearest neighbor search fast with kd-tree structure and GPUs. Moreover, our method reduces deterioration of an accuracy of approximate nearest neighbor search.

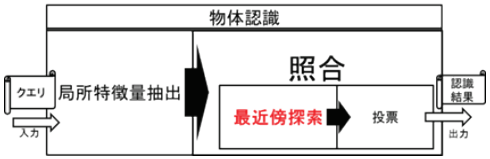

図 1　物体認識処理の流れ

1. はじめに

物体認識処理の 1 つに，物体が撮影された画像をデータベース（以下，DB）中の画像群と照合し，同じ物体の画像を特定する処理がある．本研究では，局所特徴量を用いた物体認識処理技術 1）について取り扱う．この局所特徴量を用いた物体認識処理の流れは図 1 のようになり，処理を局所特徴量抽出と照合に大別することができる．

局所特徴量抽出は，SIFT 2），SURF 3）等のアルゴリズムを用いて，1 枚の入力画像（以下，クエリ）から，数百～数千の高次元ベクトル群を抽出する．DB 中の画像群にも予め，局所特徴量抽出が行われており，物体認識処理の際は，抽出された高次元ベクトル群が利用される．

一方，照合はさらに，最近傍探索と投票の 2 つの処理に分けることができる．最近傍探索では，クエリから抽出されたベクトル（以下，クエリベクトル）群に対して，各クエリベクトルと，DB 中画像から抽出されたベクトル（以下，DB ベクトル）群とのユークリッド距離を計算する．そして，個々のクエリベクトルに対して，距離が最小となった DB ベクトルを最近傍ベクトルとして選択する．投票では，そのクエリベクトルに対して，選択した最近傍ベクトルが抽出された DB 中の画像 ID に 1 票を投票する．この一連の処理を，各クエリベクトルに対して繰り返す．最終的に，最多得票数を得た画像 ID が，撮影した物体の認識結果として出力される．

この局所特徴量を用いた物体認識処理の高速化を考えた場合，局所特徴量抽出については，高速なアルゴリズム（SURF）や GPU を利用した SIFT-GPU 4）などにより高速化が実現されている．しかし，照合処理中の最近傍探索においては，照合対象となる DB 中の画像群が増大した際，DB ベクトルの数も増えるため，処理コストが増大するという問題がある．一方，投票に要する処理時間は，事前実験から局所特徴量抽出及び照合の最近傍探索に比べて十分に小さいことが確認された．従って，最近傍探索の高速化を実現することは，多数の物体を認識対象とする高速な物体認識処理につながると期待できる．そこで本研究では，大規模な高次元ベクトル群に対する高速な最近傍探索について検討を行う．そして，少なくとも，1000 個のクエリベクトル群に

[†]NTT サイバースペース研究所
NTT Cyber Space Laboratories

Vol.2012-DPS-150 No.24
Vol.2012-CSEC-56 No.24
2012/2/29

対して，DB ベクトルが 1 億個の場合の最近傍探索処理を 1 秒以内に処理することを目指す．

2. 従来技術と課題

2.1 従来技術

DB 中の画像群が大規模化すると，DB ベクトル群が大規模化する．大規模化した DB ベクトル群を想定し，クエリベクトル数を 1000 個，DB のベクトル数を 500 万個とした場合で全探索を行う予備実験を行った結果，3.4GHz の CPU のコア 6 つで並列に処理しても，約 400 秒の処理時間がかかった．そのため，近傍探索を高速化する従来技術として，探索範囲を絞ることで高速化を図る近似最近傍探索と，GPU（Graphics Processing Unit）の並列演算能力の高さを利用した全探索の並列高速化技術がある．

2.1.1 近似近傍探索（kd-tree based ANN）

大規模なベクトル群に対して，高速な探索を行う手法の 1 つに，近似最近傍探索 5）（Approximate Nearest Neighbor search，ANN）がある．全探索のような厳密な探索は，真の最近傍ベクトルを求めることができる．一方，近似最近傍探索は，探索範囲を絞ることで，真の最近傍ベクトルは得られない可能性がでてくるが，距離が近いであろうベクトルを高速に探索することが可能な手法である．探索範囲を絞る手法のため，探索範囲を狭くすれば，探索精度は低下するが，探索速度は速くなる．その逆に探索範囲を広くすれば，探索精度は向上するが，探索速度は遅くなる．

kd-tree based ANN 6）は，kd-tree 7）によるデータ構造を用いた近似最近傍探索である．kd-tree は，図 2 のように，特定の次元に着目して空間を分割することで，木を構成する．構成された木の葉ノードは，分割された空間が対応する（図 2）．探索では，kd-tree の探索アルゴリズムに従って，クエリベクトルが葉ノードに到達すると，到達した葉ノードに対応付けられている DB ベクトルと距離計算を行い，その計算結果を暫定解 r として保持する．そして，暫定解 r を用いて，バックトラック処理を行う（図 3）．バックトラックではまず，クエリを中心とした半径が暫定解 r の超球を描く．次に，その超球と交わる範囲の葉ノードに対応した DB ベクトルと距離計算を行い，最小の距離となる DB ベクトルを結果として出力する．バックトラックを行うことで，最初の探索で到達した葉ノードに対応する DB ベクトルよりも距離が短い DB ベクトルがないかを確認できる．この手法を，本研究でターゲットしているようなベクトルの次元数が高く，DB ベクトルの数が大規模な場合，半径 r のままで処理を実行すると，莫大な処理時間がかかる．そこで，この半径 r に，バックトラック半径度合 α（$0<\alpha<1$）を用いて，半径を狭める．この狭めた半径を，近似半径r'とすると，

$$r' = \alpha \cdot r \quad (r' < r)$$

となる．近似半径を用いると，クエリを中心とした半径r'の超球と交わる範囲内の葉

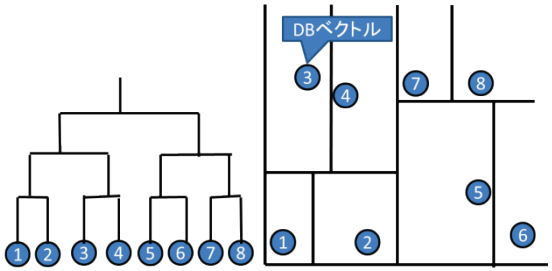

図 2　kd-tree による構造化

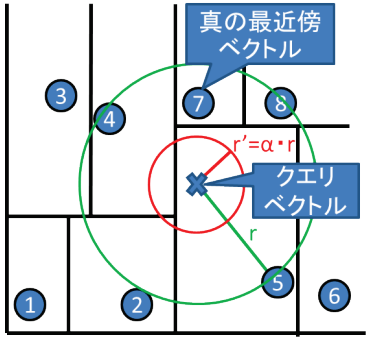

図 3　バックトラック概略図

ノードの数は，半径 r の時に対して減るので，計算対象の DB ベクトル数が減り高速化に繋がる．しかし，必ずしも厳密な解が求まるとは限らない．例えば，図 3 では，バックトラック半径度合 α を，暫定解 r に掛けることで，厳密に探索するなら計算対象に入れるべき葉ノードが含まれなくなるので，得られる解は必ずしも厳密解とは限らない．

2.1.2 GPU を用いた並列化全探索

最近傍探索を高速化するもう 1 つの従来技術として，GPU を利用したベクトル間のユークリッド距離計算の大規模並列化がある．最近傍探索処理は，ベクトル間のユークリッド距離計算を行う段階と，計算結果の集合から最小値である最短距離に位置するベクトル(最近傍ベクトル)を求める段階の 2 つのステップに分けることができる．この内，ユークリッド距離計算の計算コストが，最短距離を求めるコストに比べて圧倒的に大きい．事前実験において，クエリベクトル数を 1000 個，DB のベクトル数を

Vol.2012-DPS-150 No.24
Vol.2012-CSEC-56 No.24
2012/2/29

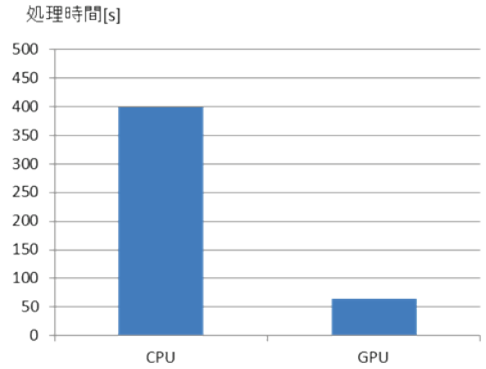

図 4　500 万ベクトルを全探索時の処理時間の比較

500 万個，ベクトルの次元数 128 次元のベクトル群に対して，CPU を用いて全探索による最近傍探索を行ったところ，ユークリッド距離計算に約 400 秒かかり，最近傍ベクトルを選択する処理には約 30 ミリ秒しかかからなかった．そのため，ユークリッド距離計算の段階を高速化することは，最近傍探索処理全体の高速化に繋がる．

このユークリッド距離計算を大規模並列化する手法として，GPU を利用する．GPU は SIMD（Single Instruction Multiple Data）命令を得意としているため，本研究においては，各クエリベクトルと DB ベクトル群との距離計算を SIMD 命令 によって並列化する．この並列化によって，大規模なベクトル群に対しても高速な処理が期待できる．

また，今回想定する 128 次元のベクトル間のユークリッド距離 D は，クエリベクトルの各要素をq_i，DB ベクトルの各要素をd_iとすると，

$$D = \sqrt{\sum_{i=1}^{128}(q_i - d_i)^2}$$

で表すことができる．この時，各要素の減算の 2 乗値を 128 回足すことで，距離 D の 2 乗値を得ることができる．本研究では，距離計算結果を大小比較することに用いるため，最後の平方根計算は必要ない．各要素の減算の 2 乗値間の加算は，並列処理に適しているリダクション 8）手法を用いることで高速化が期待でき，それによって各ユークリッド距離計算自体の高速化が期待できる．従って，GPU を用いることで，ユークリッド距離計算の並列化により，大規模なベクトル群に対して，最近傍探索の高速化が期待できる．

表 1　本研究で用いた CPU と GPU の構成

	ホスト	GPUデバイス
processor	Intel® Xeon® X5650 6 Cores @ 2.67GHz	NVIDIA Tesla M2090 512 CUDA Cores @ 1.3 GHz
memory	24GB	5.6GB (グローバルメモリ)

事前実験として，500 万ベクトルを全探索した場合の探索時間を測定した．測定は，CPU と GPU で探索した場合のぞれぞれで行った．表 1 のスペックの CPU と GPU による測定結果を図 4 に示す．CPU で探索した場合よりも GPU で探索した方が高速であることは確認できた．しかし，単純に全探索するだけでは，GPU を用いたとしても近似近傍探索における処理速度に対して，十分な速度向上は得られなかった．

2.2 課題

近似最近傍探索を用いた場合，真の最近傍ベクトルを求めることができる精度を探索精度とすると，探索精度と探索速度はトレードオフの関係にある．また，探索精度が高いと，物体認識処理の出力結果である認識精度も高くなるという報告がなされている 9)．探索速度を上げるには，探索精度を下げなければならず，結果として，認識精度の低下を招く．

一方，GPU を用いた並列化全探索手法では，全探索の高速化を試みているため，探索精度は 100%を維持できる．しかし，事前実験からも，十分な処理速度の向上は得られない．

本研究では，物体認識の出力結果の精度である認識精度を高く保った高速な探索処理を実現したい．そのためにも，探索精度の低下を抑えた高速な探索手法を検討する必要がある．

3. 提案手法

3.1 着眼点

探索精度の低下を抑えた高速な探索を実現するために，前章で述べた 2 つの従来手法の特徴に着眼した．

まず，近似最近傍探索については，探索範囲を絞る点に着眼した．近似度合を大きくし，探索範囲を絞ることで，探索精度は低下するが，距離計算の対象となるベクトル数が減るので，探索速度は速くなる．

GPU を用いた全探索の並列化については，ユークリッド距離計算を大規模に並列処

Vol.2012-DPS-150 No.24
Vol.2012-CSEC-56 No.24
2012/2/29

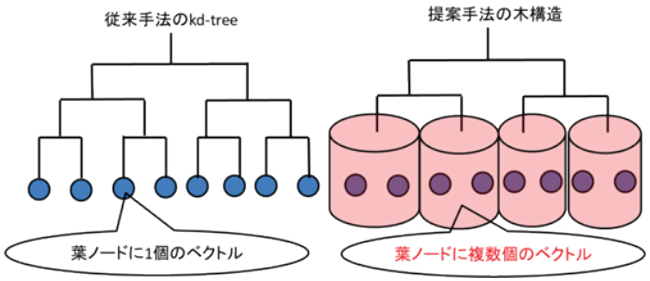

図 5　葉ノードに複数ベクトルがある木構造

理可能な点に着眼した．本研究で想定しているような大規模なベクトル群に対して，GPU による距離計算の大規模並列化は，単位時間当たりに処理できるベクトル数を増加させることができるので，探索速度が速くなる．

これら 2 つの着眼点から，探索範囲は絞るが，GPU の計算能力を活用し，広範囲の DB ベクトルを高速に探索できる手法を提案する．探索範囲を広くできることで，距離計算できる DB ベクトル数が増え，結果として探索精度も高く保つことができる．本提案手法を，GPU を用いた構造化データ高速探索手法と呼ぶこととする．

3.2　手法の詳細

GPU を用いた構造化データ探索手法は，例えば従来手法の kd-tree を用いて，インデキシングによるデータ構造化を施し，葉ノードに複数の DB ベクトルが対応づけられるように DB ベクトル群を分割する．そして，各葉ノードに対応する複数の DB ベクトルを GPU で並列計算する手法である．従来の近似最近傍探索では，分割した領域に，1 つもしくは少数の DB ベクトルが保持されていた．しかし，本提案手法では，GPU の高い並列計算能力を活かすことを考慮して，分割した領域に多数の DB ベクトルを保持させることを提案する（図 5）．そのため，複数の DB ベクトルと高速に並列計算できるので，真の最近傍ベクトルにあたる確率を高く維持した高速な探索が可能になる．以下で，GPU の処理に適した具体的なデータ構造化手法を示す．

データの構造化は，クエリベクトル群が入力される前の事前処理として，CPU 処理によって行われる．本研究では，従来技術でも用いられている kd-tree による木構造を，GPU を用いた並列探索処理に適した形態にカスタマイズする．kd-tree は，特定の次元に着目し，その次元の特定値（例えば，各ベクトルの特定次元の値から得られる中央値）の超平面で高次元の空間を 2 分割していくことで，ベクトル群を構造化する．従来手法では，分割処理が終了した時点で，葉ノードに対応するベクトルが 1 つになるまで，空間が分割される．しかし，本研究では，GPU を用いることで，複数のベクトル群を並列処理することを目的に，分割の最終段階で，葉ノードに 1 つのベクトルではなく，複数のベクトルが対応するようにする（図 5）．そのために，空間の 分割を途中で止める手法を用いる．分割を途中で止めることで，構造化処理の最終的な分割空間には，1 つではなく複数のベクトルが対応することになる．この時，分割を止める木の深さをパラメータとして調整することで，最終的な葉ノード数を設定することができる．構造化されたデータは，木構造の左端ノードから 右端ノードに向かって，順に 1 次元配列上に保持される．この構造化処理は，CPU 側（以下，ホスト側）で処理され，処理結果の 1 次元配列を GPU デバイス側のグローバルメモリにコピーすることで，構造化データが GPU 側でも処理できるようになる．

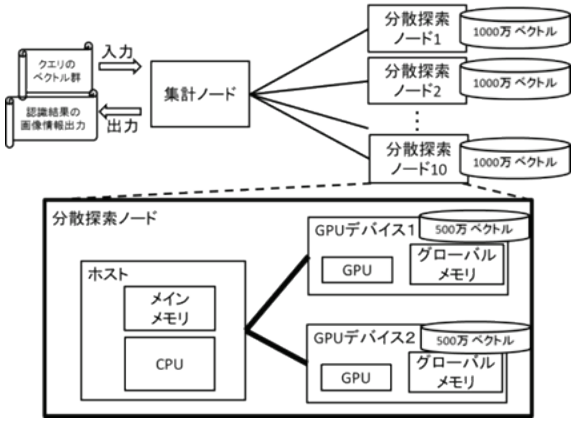

図 6　全体構成（上）と分散探索ノード構成（下）

4.　実装

4.1　構成

4.1.1 システム全体の構成

前章の提案手法を実現する実装方法を提案する．提案するシステムの構成を図 6（上図）に示す．構成は 1 台の集計ノードと複数台の分散探索ノードから成る．各分散探索ノードのマシン構成を表 1 に示す．GPU で最近傍探索の並列処理を高速化するには，GPU のグローバルメモリ上に DB ベクトル群を保持させておく必要がある．今後 DB ベクトル数が増大することを想定した場合，1 台の GPU で高速に処理させることは，グローバルメモリ容量の観点からも難しいと考えられる．従って，GPU を搭載したマ

Vol.2012-DPS-150 No.24
Vol.2012-CSEC-56 No.24
2012/2/29

シンを複数台用意し，DB ベクトル群を水平分割する．これによって，大規模なベクトル群に対する最近傍探索処理の GPU による高速並列化探索が可能になる．

本研究では，大規模な DB ベクトル群として，1 億 240 万ベクトルをシステム全体で取り扱った．1 億 240 万ベクトルを 10 台の分散探索ノードに水平分割し，分散探索ノード 1 台あたり，1024 万ベクトルずつ保持させた．以上が実装システム全体の構成であり，以下で分散探索ノードの構成について述べる．

4.1.2 分散探索ノードの構成

分散探索ノードの構成を図 6（下図）に示す．分散探索ノードは，CPU やメインメモリを搭載したホスト側と，GPU を搭載した 2 台の GPU デバイス側から構成される．そして，GPU デバイス 1 台あたり，512 万 DB ベクトルを保持することで，分散探索ノードあたりでみた時，1024 万 DB ベクトルを保持している．

4.2 実装システムの処理フロー

事前処理として，各分散探索ノードにおいて，総計 1024 万ベクトルを 512 万ベクトルずつに 2 分割し，それぞれの 512 万ベクトルに対して，ホスト側で kd-tree による木構造化を行う．構造化されたデータを GPU デバイス側のグローバルメモリにコピーする．ホスト側，GPU デバイス側の両方で構造化されたデータが保持された状態で，クエリベクトルを待ち受ける．

クエリベクトル群が実装システムに入力された時の処理フローを述べる．

集計ノード：

1. クエリベクトル群の入力を受け付ける．
2. 分散探索ノードへクエリベクトル群を送信する．

各分散探索ノード：

3. ホスト側で，クエリベクトル群を受信する．
4. ホスト側で， 2 つの木構造化された DB 側のベクトル群に対して ，kd-tree の探索アルゴリズムに従って，各クエリベクトルは，葉ノードに到達する．
5. ホスト側で，各クエリベクトルが到達した葉ノードの ID とクエリベクトル群を GPU デバイス側に転送する．
6. 各 GPU デバイス側で，転送されてきたクエリベクトル群に対して，それぞれのクエリベクトルが到達した葉ノード内の DB ベクトル群とユークリッド距離の並列計算を行う．
7. 各 GPU デバイス側からホスト側に，探索した葉ノード中での最短距離結果を転送する．
8. ホスト側で，転送されてきた葉ノード中の最短距離値とバックトラック半径度合 α （$0<\alpha<1$）を用いて，バックトラック処理を行う．
9. ホスト側から各 GPU デバイス側へ，バックトラック対象となった葉ノード群を転送する．
10. 各 GPU デバイス側で，バックトラック対象の葉ノードに対応する D B ベクトル群について最近傍探索を行う．
11. 各 GPU デバイス側で，各クエリベクトルについて，木構造の探索時に到達した葉ノードとバックトラック対象となった葉ノード群から，最近傍となるベクトルを求める．
12. 各 GPU デバイス側からホスト側へ，それぞれの GPU デバイスが扱う DB ベクトル群中での最近傍ベクトルを転送する．
13. ホスト側で，2 台の GPU デバイスから送られてきた各 GPU デバイス中での最近傍ベクトルから，分散探索ノード単位で見たとき最近傍ベクトルを選択し，選択結果を集計ノードに送信する．

集計ノード：

14. 各分散探索ノードからの処理結果の受け取る．
15. DB 全体での最近傍ベクトルを選択する．
16. 投票を行い，最多得票の画像 ID を出力する．

5. 実験・考察

4 章で述べた実装方法にしたがって，実験システムを実装し，評価実験を行った．本提案手法の有効性を検証するため，従来手法との比較実験を 3 つ行った．比較に用いた従来手法は，kd-tree based ANN である．

まず，3 つの実験の関連性を示す．本提案手法におけるパラメータとして，葉ノードのサイズを変更するパラメータと，バックトラック半径度合を決めるパラメータの 2 つがある．最初の 2 つの実験では，そのどちらか一方を固定して，もう一方を変化させた時の探索時間についての測定を行った．これによって，各パラメータ変化時の探索時間の変化傾向を検証した．そして，最後の 3 つ目の実験で，これら 2 つのパラメータを両方変化させた時の探索時間と精度の関係を測定することで，従来手法と比較して，高い精度を維持した高速な探索ができているかを検証した．本実験における探索精度とは，厳密な探索である全探索時の結果を 100%とし，全クエリベクトルの中で，全探索時と同じ最近傍ベクトルを求めることができたクエリベクトル数を，クエリベクトルの総数で割った値とする．

Vol.2012-DPS-150 No.24
Vol.2012-CSEC-56 No.24
2012/2/29

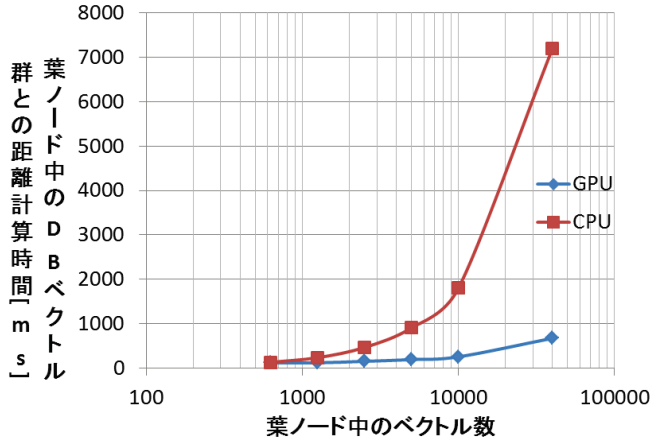

図 7　葉ノード中のベクトル数を変化させた時の処理速度比較

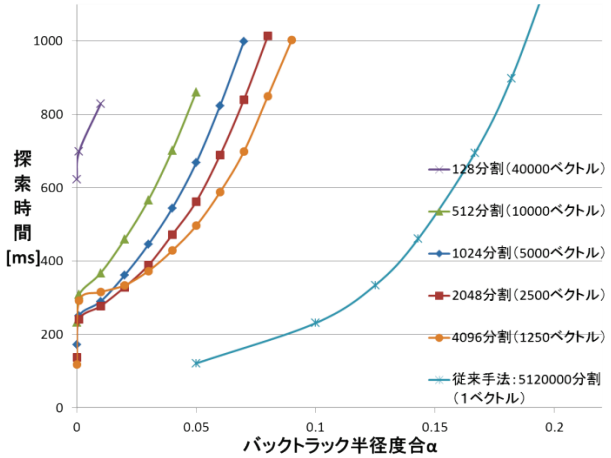

図 8　バックトラック半径度合 α の変化に対する探索時間

5.1 葉ノードサイズに対する探索時間の測定

5.1.1 目的・方法

本研究では，各分散探索ノードにおける 512 万個の DB ベクトル群を分割することで，探索範囲を削減する．そして，分割した範囲を GPU で並列に探索することで，高速化を図っている．しかし，分割数を増やし，葉ノードあたりのベクトル数がある値以下だと，GPU で並列処理を行う場合よりも，CPU で行った場合の方が高速であると推測できる．例えば，分割数を 256 万分割として，葉ノード当たりのベクトル数が 2 ベクトルの場合，これら 2 ベクトルの並列処理は，GPU で行うより，CPU で行った方が高速であると考えられる．このことを確認するために，実験を行う．実験は，分割数を変えて，葉ノード中のベクトル数を変化させた時の 1 つの葉ノードの並列探索時間について測定を行う．測定は，1 つの葉ノード中のベクトル数を，40000，10000，5000，2500，1250，625 に変化させた場合の CPU と GPU それぞれの探索処理時間について行われる．CPU での探索処理時間は，図 6（下図）のホストのみが稼動する構成で測定され，GPU での探索処理時間は，図 6（下図）のホストと 1 つの GPU デバイスが稼働する構成で測定される．比較に用いた CPU と GPU の仕様は，表 1 のものである．

5.1.2 結果

測定結果は，図 7 のようになった．分割数が少なく，葉ノード中のベクトル数が多いと，GPU による並列処理の方が，CPU による並列処理よりも優位である．分割数が多く，葉ノード中のベクトル数が少なくなるにつれ，GPU と CPU 間の処理時間の差が縮まる．実験では，葉ノード中のベクトル数が 625 個の場合に，GPU と CPU の探索時間がほぼ同じになった．以上の結果より，本実験においては，葉ノード中のベクトル数が 1250 個以上の場合，GPU による並列探索処理の優位性が確認できる結果となった．

5.1.3 考察

分割数を多くすると葉ノード中のベクトル数が減り，CPU を用いた並列処理の方が，GPU を用いた並列処理より高速になる傾向を示した．この原因として，CPU のホスト側-GPU デバイス側間の転送時間が関係していると考えられる．GPU で並列計算を行った場合，計算前にクエリベクトル群を CPU ホスト側から GPU デバイス側に転送し，計算後は計算結果を GPU デバイス側から CPU ホスト側に転送しなければならない．従って，クエリベクトル数が増加すると，転送コストも増加する．本研究では，クエリベクトル数は，1000 ベクトルとして検討を行ったので，転送コストは一定である．このため，葉ノード中のベクトル数が減り，探索処理時間が短くなると，転送コストの影響が大きくなってくる．本実験の結果から，葉ノードのサイズに応じて CPU を用いるか，GPU を用いるかを選択する必要があることが示された．

Vol.2012-DPS-150 No.24
Vol.2012-CSEC-56 No.24
2012/2/29

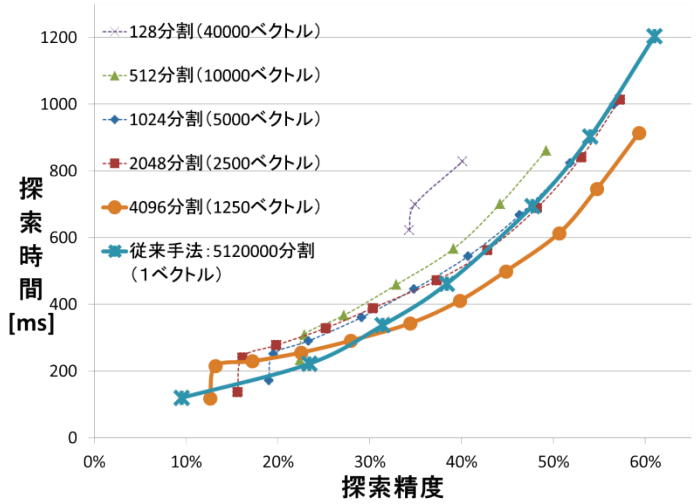

図 9　葉ノードのベクトル数とバックトラック半径度合 α をパラメータとした時の速度と精度の測定

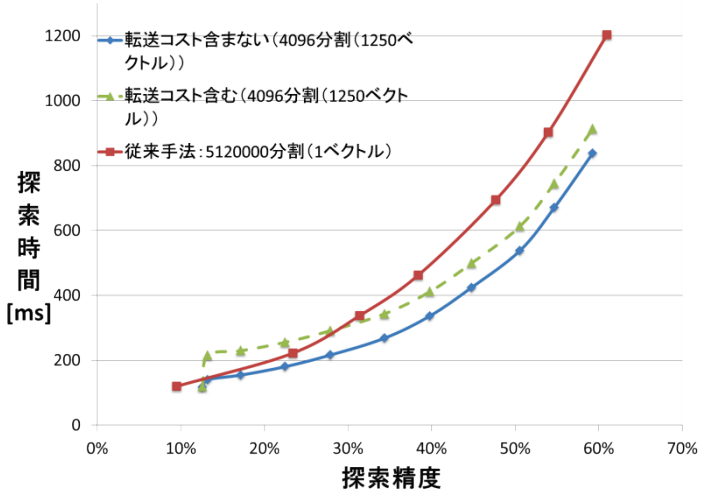

図 10　転送コストを除いた時の従来手法との比較

5.2 バックトラック半径度合に対する探索時間

5.2.1 目的・方法

従来手法と同様に，バックトラック半径度合 α の値が大きいと，探索の厳密さは増すが，探索速度は遅くなると予想される．このことを確認するために，本実験を行う．実験は，葉ノード中のベクトル数を固定した状態で，バックトラック半径を 0 から変化させた時の探索時間を測定する．

5.2.2 結果

結果を図 8 に示す．図 8 より，提案手法，従来手法の両方において，どの葉ノードサイズにおいても，バックトラック半径度合 α の値が大きくなると，探索処理時間が大きくなった．

5.2.3 考察

この結果は，バックトラック半径度合 α の値が大きいと，探索の厳密さが増す反面，探索速度の低下を招くという予想と一致している．

一般に，バックトラック半径度合 α が大きいと，探索の厳密さが増すので，探索精度も向上する．しかし，本実験のように，葉ノードサイズを変えてみた場合，バックトラック半径度合 α を大きくした時の探索速度の上昇率に対して，探索精度の上昇率が一定とは限らない．そこで，葉ノードサイズとバックトラック半径度合 α を変化させた時の探索時間と探索精度に関して測定を行った．

5.3 葉ノードサイズと α が変化した時の速度と精度

5.3.1 目的・方法

本提案手法におけるパラメータである葉ノードサイズとバックトラック半径度合 α の 2 つを変化させた時に，探索時間と探索精度がどう変化するかを測定する．これによって，精度固定で結果をみた場合，従来手法に対して，本提案手法がどの程度高速になっているかを検証する．実験は，1 つの葉ノード中のベクトル数が，40000，10000，5000，2500，1250 のそれぞれの場合で，バックトラック半径度合 α を 0 から大きくしていった時の探索精度と探索時間を測定する．

5.3.2 結果

測定結果を図 9 に示す．図 9 より，精度一定でみたときに，探索精度 30%以上において，従来手法の kd-tree より高速であることを確認した．例えば，探索時間 800 ms でみた時の提案手法の精度約 50%においては，約 20%の高速化が確認できた．また，本提案手法が速度面で優位な探索精度 30%以上においては，葉ノード中のベクトル数が 1250 個の場合が，最も高速であった．

Vol.2012-DPS-150 No.24
Vol.2012-CSEC-56 No.24
2012/2/29

5.3.3 考察

本実験によって，一定以上の探索精度において，従来手法よりも高速であることが確認できた．しかし，本手法は，さらに高速化できる可能性がある．それは， バックトラックによって生じる CPU－GPU 間の転送コストを他の処理と重ねて実行することで，転送処理を処理時間上からは見えないようにする方法である．葉ノードの DB ベクトル数が 1250 個の場合で, バックトラックを行った時の転送コストは約 70 ms であった．そのため，この転送を他の処理と重ねることができれば，探索時間は図 10 のようになり，本実験結果よりも高速な処理が可能になると考えられる．

6. まとめと今後の課題

本研究では，GPU を用いた構造化データ高速探索手法を提案した．提案手法と従来手法の比較を行った結果，30 %以上の探索精度において，探索精度一定でみたときに従来手法に比べて，高速であることが確認できた．例えば，探索時間を 800ms でみた時の提案手法の精度では，従来手法に対して，約 20 %の高速化が確認できた．

今後の課題としては，スループット向上の検討が主に考えられる．

参考文献

1) 藤吉弘亘，山下隆義，-CVIM チュートリアルシリーズ- コンピュータビジョン最先端ガイド 2，八木康史（編），斉藤英雄（編），pp.1-58，アドコム・メディア株式会社，東京，2010.
2) D. Lowe, Disrinctive Image Features from Scale-Invariant Keypoints, International Journal of Computer Vision, vol.60, no.2, pp.91-110, Jan.2004.
3) H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, SURF: Speeded Up Robust Features, Computer Vision and Image Understanding, vol.110, no.3, pp.346-359, June.2008.
4) S. N. Shiha, J-M. Frahm, M. Pollefeys, and Y. Genc, GPU-based Video Feature Tracking And Matching, In Workshop on Edge Computing Using New Commodity Architectures, May.2006.
5) 和田俊和，“最近傍探索の理論とアルゴリズム”，2009 情報処理学会 CVIM 研究会，vol.2009-CVIM-169, no.12, pp.1-12, Nov.2009.
6) S. Arya, D. M. Mount, R. Silverman and A. Y. Wu, An optimal algorithm for approximate nearest neighbor searching, Journal of the ACM, vol.45, no.6, pp.891-923, Nov.1998.
7) J. L. Bentley, Multidimensional binary search trees used for associative searching, Communications of the ACM, vol.18, no.9, pp.509-517, Sept.1975.
8) 青木尊之，額田彰，はじめての CUDA プログラミング，第二 I/O 編集部（編），pp.134-143，株式会社工学社，東京，2009.
9) 藤吉弘亘, 山下隆義, -CVIM チュートリアルシリーズ-コンピュータビジョン最先端ガイド 3，八木康史（編），斉藤英雄（編），pp.79-82，アドコム・メディア株式会社，東京，2010.