DEIM Forum 2022 K24-1

GPUを用いた高次元データに対する逆k最近傍検索の高速化手法の改善

対比地恭平† 天笠 俊之††

† 筑波大学 理工情報生命学術院 システム情報工学研究群 〒305–8577 茨城県つくば市天王台 1–1–1
†† 筑波大学 計算科学研究センター 〒305–8577 茨城県つくば市天王台 1–1–1
E-mail: †tsuihiji@kde.cs.tsukuba.ac.jp, ††amagasa@cs.tsukuba.ac.jp

あらまし 逆k最近傍検索とは，あるクエリ点が与えられたときに，それをk最近傍に持つようなすべてのデータ点を検索するものである．逆k最近傍検索は意思決定支援システムや地理情報システム (GIS)，外れ値検出など，幅広い分野で応用されており，近年注目を集めている．しかし，既存の研究は低次元のデータや小規模なデータのみに対応しているものが多く，高次元のデータや大規模なデータを扱えないという問題がある．そこで我々は高次元の大規模データを対象とし，GPU を用いた逆k最近傍検索の高速化手法を提案したが，この手法では前処理が必要であり，データが大規模になるほど膨大な計算コストがかかってしまうという問題があった．本研究では，今まで前処理として扱っていた計算部分も含めて，高速化を図る．

キーワード 逆k最近傍検索，GPU，高次元データ

1 はじめに

近年，情報科学の分野で用いられるデータは増加し続けている．これらのデータを解析する際には膨大なコストがかかるため，効率的に処理する手法が求められている．データ解析に用いられる手法の一つに，逆k最近傍検索 [1] がある．逆k最近傍検索とは，あるクエリ点が与えられたときに，それをk最近傍に持つようなすべてのデータ点を検索する処理である．逆k最近傍検索は意思決定支援システムや地理情報システム（GIS），外れ値検出など，幅広い分野で応用されており [1]，近年様々な手法が提案されている．

逆k最近傍検索の従来手法の多くは，逆k最近傍検索をフィルタリング段階と検証段階の二つに分けた，フィルタリング・検証アプローチを採用している．フィルタリング段階では，空間分割などによって解の候補を絞り込むとともに，枝刈り戦略を組み合わせることで，効率的に解の候補を探索する．ただし，候補解には真の解ではないものも含まれる．検証段階では，フィルタリング段階で残ったデータ点とクエリ点の距離を計算し，クエリ点が逆k最近傍かどうかを検証する．このように二段階に分けることにより，効率的な解の探索を実現している．

先述以外の逆k最近傍検索の応用先として，アカウントベースドマーケティング（ABM）が挙げられる．ある商品をクエリとして逆k最近傍検索を行うことで，それに関心のあるユーザアカウントを検出する．企業はその結果を元にターゲットを絞り，新しいサービスを提供することができる．このとき，商品やユーザはいずれも高次元の特徴ベクトルとして扱われる．しかし，高次元空間における逆k最近傍検索は，枝刈りによるコスト削減が難しく，次元の呪いによって計算コストが大きくなってしまう．

一方，本来グラフィックス処理に使われる GPU（Graphics Processing Units）を汎用計算に用いる GPGPU（General-Purpose Computing on GPU）が近年注目を集めている [2]．GPU は CPU と比べて小規模なプロセッサを大量に搭載するため，高い並列処理性能を持つ．ゆえに，GPU を効果的に利用することで，大規模な演算を高速に行うことができる．ただし，GPU の構造は特殊であるため，利用する際にはアルゴリズムを最適化する必要がある．

従来の手法はいずれも CPU による逐次処理のみに対応しており，GPU による並列処理を用いた手法は提案されていない．そこで我々は，GPU を用いた逆k最近傍検索の高速化手法を提案した [3]．この手法は，ナイーブな逆k最近傍検索を前処理と問合せ処理に分け，問合せ処理を並列化し，GPU によって高速に処理するものだ．高次元のデータに対しても高速化に成功しており，メモリ効率の面でも高い性能を示すが，データ数が増えるにつれて前処理の計算コストが膨大になってしまうという問題がある．

そこで本研究では，我々の先行研究の改善手法を提案する．具体的には，前処理とされていた，各データ点のk最近傍までの距離の計算を GPU によって高速化し，前処理が不要な逆k最近傍検索の高速化を目指す．

本稿の構成は，以下の通りである．2 節で逆k最近傍検索，3 節で GPGPU について説明する．4 節で関連研究を紹介し，5 節で提案手法について述べ，6 節で実験結果を示す．最後に 7 節で本稿のまとめを述べる．

2 逆k最近傍検索

データ解析手法の一つに，逆k最近傍検索 [1] がある．逆k最近傍検索とは，あるクエリ点が与えられたときに，それをk最近傍（最も近いk個の点）に含むようなすべてのデータ点を見つける処理のことである．すなわち，クエリ点qが与えられたとき，データセットSについて以下の式で定義される．

$$RkNN(q) = \{p | p \in S \wedge \kappa_p \leqq k\}$$

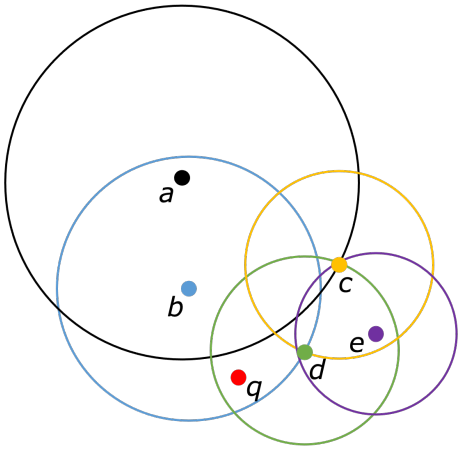

図 1: $k = 2$ のときの逆 k 最近傍検索

ここで，κ_p は，q から p よりも近い位置にある点の数を意味し，以下の式で定義される．

$$\kappa_p = |\{p' | p' \in \{S - \{p\}\} \wedge d(p', p) \leqq d(p', q)\}|$$

ここで，$d(\cdot)$ は距離関数を表しており，一般的にユークリッド距離が用いられる．

図 1 は $k = 2$ のときの逆 k 最近傍検索の例を示しており，点 a, b, c, d, e はデータ点，点 q はクエリ点を表している．この例において，b と d が 2 最近傍までの距離の内側に q を含むため，q の逆 2 最近傍となる．

逆 k 最近傍検索の別の応用先として，アカウントベースドマーケティング（ABM）が挙げられる．ある商品をクエリとして逆 k 最近傍検索を行うことで，それに関心のあるユーザアカウントが検出される．企業はその結果を元にターゲットを絞り，新しいサービスを提供することができる．このとき，商品やユーザはいずれも高次元の特徴ベクトルとして扱われる．しかし，高次元空間における逆 k 最近傍検索は，次元の呪いによって計算コストが大きくなってしまう．

3 GPGPU

GPU（Graphics Processing Units）はグラフィックス処理に特化した演算装置である．逐次的な処理を得意とする CPU（Central Processing Units）に対し，GPU は簡単な処理の並列実行に優れる．そのため，GPU を汎用計算に用いる技術が近年注目されており，GPGPU（General Purpose computing on GPU）と呼ばれる [2]．GPU を効果的に使うことで，従来には得られなかった演算性能を達成することが可能となる．

GPGPU は，基本的に以下の流れで実行される．

(1) GPU 上で扱うデータのためのメモリを確保する
(2) データを GPU 上に転送する
(3) カーネル（GPU 上の処理を定義した関数）を起動する
(4) GPU による処理
(5) 処理後のデータを CPU に転送する

ここで，CPU-GPU 間のデータ転送は低速であるため，データ転送やカーネル起動の回数はなるべく抑える必要がある．

本研究は，NVIDIA 社の GPU と CUDA [4] を用いて GPU プログラミングを行った．以下では NVIDIA 社の GPU の構造と特徴，及び CUDA を用いたプログラミングについて説明する．

3.1 NVIDIA GPU

GPU は複数のストリーミングマルチプロセッサ（SM）から成り，各 SM は数十から数百のストリーミングプロセッサ（SP）を搭載している．一つ一つの SP はクロック周波数が低く，複雑な処理には適さないが，単純で大規模な演算を行う場合，複数の SP を並列に使用することで，高速に処理することができる．

また，GPU には大きく分けて，グローバルメモリ，シェアードメモリ，レジスタの 3 種類のメモリが存在する．グローバルメモリは最も容量が大きく SM 外にあるためどの SP からもアクセスできるが，アクセス速度が遅い．シェアードメモリは同じ SM 内の SP からしかアクセスできないという制約があるが，グローバルメモリよりも高速にアクセスできる．レジスタは最も容量が小さく，同じ SM 内の SP からしかアクセスできないが，シェアードメモリよりもさらに高速なアクセスが可能である．GPU を使用する際には，各メモリの特徴を考慮してデータを扱う必要がある．

3.2 CUDA

CUDA（Compute Unified Device Architecture）[4] は NVIDIA 社が開発・提供している，GPU 向けの統合開発環境である．専用の C/C++コンパイラ（nvcc）や API が提供されており，これらを用いることで NVIDIA 社の GPU を制御することができる．

CUDA プログラミングは，GPU アーキテクチャと対応する階層構造を基軸としており，スレッド・ブロック・グリッドの 3 層で構成される．スレッド処理は各 SP で実行され，ブロック処理は各 SM で実行される．ブロックの集合をグリッドと呼び，GPU 上で行う一連の処理に当たる．実際に使用するスレッド・ブロック・グリッドの数は，処理内容に応じて適切に指定しておく必要がある．

4 関 連 研 究

逆 k 最近傍検索の従来手法の多くは，計算コストを削減するために，フィルタリング・検証アプローチを採用している．フィルタリング段階では，空間分割などによって解の候補を絞

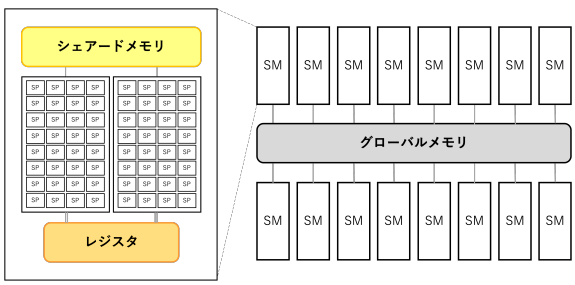

図 2: GPU アーキテクチャ

り込むとともに，枝刈り戦略を組み合わせることで，解の候補を探索する．ただし，候補解には真の解ではないものも含まれる．検証段階では，フィルタリング段階で残ったデータ点とクエリ点の距離を計算し，クエリ点が逆 k 最近傍かどうかを検証する．また，逆 k 最近傍検索の従来手法は 2 次元データのみに対応しているものが多く [5] [6] [7]，高次元データに対応した手法は少ない．

Tao らは任意の次元に対応する手法（TPL）を提案した [8] [9]．TPL は，フィルタリング段階において，クエリ点 q とデータ点 p を結んだ線分の垂直二等分線でデータ空間を分割する．二つの空間のうち，点 p 側に存在するすべての点は点 q よりも点 p の近くにあり，解にならない可能性が高いため，その領域を枝刈り候補領域とする．k 個以上の枝刈り候補領域に重複して含まれる場合，点 p は解になり得ないため，枝刈りされる．検証段階では，枝刈りされなかったデータ点から，枝刈り候補領域に含まれるデータ点やクエリ点まで距離を計算し，大小比較をすることで解を決定する．しかし，これにより効率的に検索を行えるのは低次元データのみである．

Singh らは高次元データに対応した近似逆最近傍検索手法を提案した [10]．フィルタリング段階では，クエリ点の k 最近傍を求め，それ以外の点を検索の対象から除外する．検証段階では各点に Boolean Range Query を適用し，高速に検証を行う．しかし，厳密な解は保証されていない．また，次元削減のために前処理として特異値分解をしているが，大規模データを対象とする場合，計算コストが肥大化してしまう．

我々の先行研究では，GPU を用いた逆 k 最近傍検索手法を提案した [3]．ナイーブな逆 k 最近傍検索を前処理と問合せ処理に分け，問合せ処理を並列化し，GPU によって高速に処理する．前処理ですべてのデータ点の k 最近傍までの距離を計算しておき，問合せ処理でクエリ点とすべてのデータ点の距離を計算し，前処理で求めた距離値よりも小さいものを解とする．この手法は，バッチ処理によって複数のクエリを同時に扱えるほか，高次元のデータに対しても高速化に成功しており，メモリ効率の面でも高い性能を示す．一方で，データ数が増えるにつれて前処理の計算コストが膨大になってしまうという問題がある．

5 提案手法

本研究では，我々の先行研究における前処理に GPU を用いて，ナイーブな逆 k 最近傍検索の高速化を図る．

本節では，先行研究の前処理のアルゴリズムと，GPU 上での実装方法について説明する．

5.1 前処理

前処理では，すべてのデータ点について，以下の処理を行う．なお，本手法ではユーザ定義パラメータとして，逆 k 最近傍検索を行う際に取りうる最大の k を $kmax$ と呼び，ユーザが事前に与えるものとする．

(1) 各データ点 p と p 以外のすべてのデータ点との距離を計算する

(2) 距離値を昇順にソートする

(3) 先頭の $kmax$ 個を保存する

ここで，i 番目の距離値は，点 p から i 番目に近いデータ点までの距離であり，以降では d_i と表記する．また，各データ点について $kmax$ 個の距離値を保存するのは，問合せ処理時に任意の $k(\leqq kmax)$ について逆 k 最近傍検索をできるようにするためである．

上記のアルゴリズムはナイーブな k 最近傍検索そのものである．また，データ点ごとに独立であるため，GPU の並列処理による高速化が期待できる．しかし，本研究では大規模な高次元データを対象としているため，メモリに制限のある GPU ですべての処理をまとめて扱うのは困難である．そこで，Garcia らの手法 [11] を参考にし，実装を行う．

5.2 GPU 上での実装

R と Q をそれぞれデータ点とクエリ点を列ベクトルに持つ行列としたとき，Garcia らの手法は，ユークリッド距離の計算を以下のように分解する．

$$d^2(R,Q) = N_R + N_Q - 2R^{\mathsf{T}}Q$$

ここで，N_R の i 行目は i 番目のデータ点の二乗ノルムであり，N_Q の j 列目は j 番目のクエリ点の二乗ノルムを表す．ただし，データ点とクエリ点の数をそれぞれ n_R, n_Q とすると，メモリコスト削減のため，N_R と N_Q はそれぞれ n_R, n_Q 次元のベクトルとして保持する．また，第 3 項の行列積の計算に cuBLAS [12] を使用する．cuBLAS とは，数値線形代数の基礎的演算に必要な関数を定義する API である BLAS を，CUDA で実装したものである．cuBLAS を用いることで，ベクトルや行列の演算を GPU で高速に処理することができる．距離値のソートには挿入ソートを採用する．これは，$kmax$ が小さいときに速く処理を終えることができるためである．

以上を基に，本研究では，先述の前処理アルゴリズムを以下のように実装する．

(1) CUDA によってすべてのデータ点の二乗ノルム（N_R）を計算する

(2) 全部で n 個あるデータ点（R）から，順番に n_batch 個を Q にコピーし，以下の処理を繰り返す

(2-1) CUDA によって Q の二乗ノルム（N_Q）を計算する

(2-2) cuBLAS によって $n \times n\_batch$ の行列 $A = -2R^{\mathsf{T}}Q$ を計算する

(2-3) CUDA によって A の i 行目に N_R の i 番目の要素を足し合わせる（この結果を行列 C とする）

(2-4) CUDA によって C の各列を並列にソートする

(2-5) CUDA によって C の j 列目の上位 $kmax$ 個に N_Q の j 番目の要素を足し合わせる（この結果を行列 D とする）

(2-6) D の上位 $kmax \times n\_batch$-部分行列を出力する

本手法は，GPU の空きメモリ容量に応じて n_batch を調整することで，すべての処理を GPU 上で実行する．なお，最終的に取得される $kmax \times n$ の行列は，(i,j) が j 番目のデータ

点の $d_i (i \leqq kmax)$ を表す．

6 評価実験

提案手法の性能を評価するため，本節では GPU を使用した提案手法と，同じ処理を CPU のみで行うベースライン手法について，逆 k 最近傍検索の実行時間の比較を行う．なお，ベースライン手法はシングルスレッドによる逐次処理のものとマルチスレッドによる並列処理のものを用意した．

6.1 実験環境

本実験で使用したマシンの詳細を以下に示す．

- CPU：Intel Xeon(R) Bronze 3104 @ 1.70GHz
  - OS：CentOS 7.9
  - メモリ：32GB
  - コンパイラ：g++ (GCC) 8.5.0
- GPU：NVIDIA Tesla P100
  - メモリ：12GB
  - コンパイラ：nvcc (CUDA 11.0)

実験には人工のデータセットを用いた．正規分布に従う乱数データと一様分布に従う乱数データを用意し，それぞれ次元数やデータ数を変更することができる．なお，どちらの乱数分布でも結果はほとんど変わらなかったことから，以下には正規分布に従う乱数データにおける結果のみを示す．

6.2 実験結果

6.2.1 異なるデータ数における実行時間

ベースライン手法と提案手法について，データ数を変化させたときの前処理にかかる実行時間を計測した．次元数は 256，$kmax$ は 1000 で固定し，データ数を 1 万から 200 万まで変化させた．

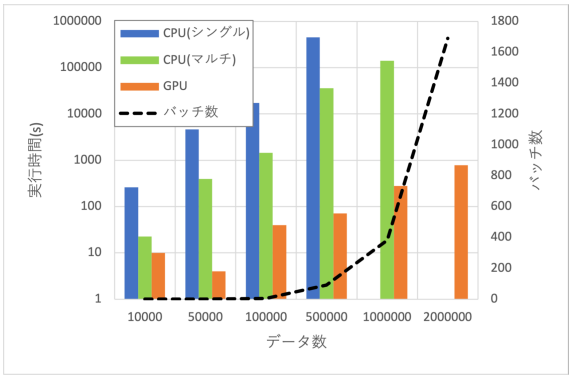

図 3: 異なるデータ数における前処理の実行時間

図 3 から，データ数がどんな数でも常に提案手法の方が高速に処理できていることがわかる．シングルスレッドのベースライン手法において，データ数が 50 万件のときの実行時間は 5 日間以上に及び，100 万件では 2 週間かけても処理が終わらなかった．また，マルチスレッドではシングルスレッドよりも平均して約 10 倍高速であったが，200 万件ではバグにより実行できなかった．一方で並列処理とバッチ処理を組み合わせた提案手法では，より大規模なデータでも高速に実行できている．さらに，提案手法は，データ数が 5 万件のときを除いて，データ数の増加に伴い実行時間も増加しているが，データ数が大きくなるほどベースライン手法との差が開くことがわかる．計測できた範囲では，データ数が 50 万件のとき，提案手法はシングルスレッドのベースライン手法の約 6425 倍高速であり，100 万件のとき，マルチスレッドのベースライン手法の約 509 倍高速であった．

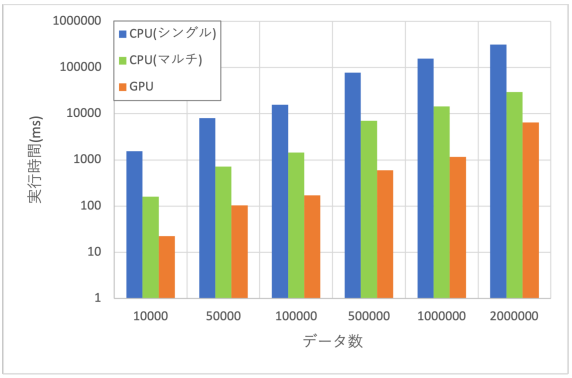

図 4: 異なるデータ数における問合せ処理の実行時間

ここで，計算結果について精査したところ，結果行列の 1 行目（同一点の距離）が 0 になっていなかった．この原因を入念に確認したところ，コード上からは問題は見当たらず，float 型に起因する誤差の蓄積によるものだと結論づけた．なお，上位 kmax 件にリストされたデータ点の ID の一致度を，真の値（ベースライン手法）と提案手法で比較したところ，多少の順位の入れ替えを除いて高い精度で一致していることを確認した．この誤差への対応は今後の課題である．

次に，本研究の提案手法によってより大規模なデータにおいて問合せ処理での性能も評価できるようになったため，ベースライン手法と提案手法について，問合せ処理にかかる実行時間を計測した．条件は上記の前処理のときと同じである．

図 4 から，問合せ処理においても，常に提案手法の方が高速に処理できていることがわかる．なお，データ数が 200 万件のときでもメモリ消費量は約 2.4GB であったことから，もっと大規模なデータにも対応することができると考えられる．

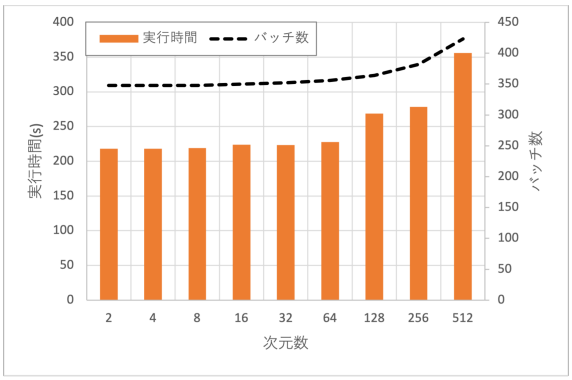

図 5: 異なる次元数における前処理の実行時間

6.2.2 異なる次元数における実行時間

提案手法について，異なる次元数における実行時間を計測した．データ数は 100 万，$kmax$ は 1000 に固定し，次元数を 2 から 512 まで変化させた．

図 5 から，64 次元までの場合にはほとんど変化がなく，128 次元以上になると実行時間が増えることがわかる．しかし，データ数の変化と比較すると，実行時間への影響は小さい．

6.2.3 異なる $kmax$ における実行時間

提案手法について，異なる $kmax$ における実行時間を比較した．データ数は 100 万，次元数は 256 に固定し，$kmax$ を 10 から 1 万まで変化させた．

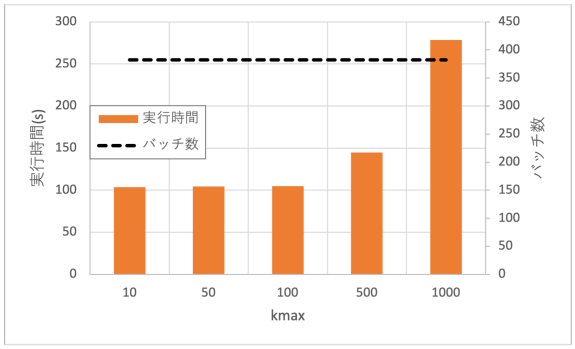

図 6: 異なる $kmax$ における前処理の実行時間

図 6 から，$kmax$ が 10 から 100 にかけては変化がほとんどなく，それ以降急激に実行時間が増加することがわかる．これは，挿入ソートの計算量が増加してしまうためだと考えられる．

7 まとめ

本研究では，高次元データを対象とする逆 k 最近傍検索を高速化するため，先行研究の前処理を GPU を用いて並列に行う手法を提案した．GPU による並列処理とバッチ処理を組み合わせることで，CPU だけでは扱えないような大規模なデータにおいて，実用的な実行時間での処理を実現した．計測できた範囲では，GPU を用いた提案手法は CPU のみのベースライン手法に比べて，シングルスレッドに対して最大約 6425 倍，マルチスレッドに対して最大約 509 倍の高速化を達成した．一方で，より大規模なデータでは，GPU 上のメモリが圧迫され，一度に計算できる数が減る代わりにバッチ数が増えてしまい，効果が薄れてしまう可能性がある．

今後は，float 型に起因する誤差を減らすための工夫を考える．また，時間のかかる挿入ソートにおいて，各列に 1 スレッドではなく複数スレッドを割り当てて並列化したり，問合せ処理も合わせて全ての処理を GPU 上で行ったりして，更なる高速化を目指す．

謝　辞

本研究は，国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務（JPNP20006）および SKY（株）共同研究経費（CPI03086）の支援によるものです．

文　献

[1] F. Korn and S. Muthukrishnan. Influenced sets based on reverse nearest neighbor queries, SIGMOD Rec., vol.29, no.2, pp.201-212, May 2000.

[2] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, GPU Computing, Proceedings of the IEEE, vol.96, no.5, pp.879–899, May 2008.

[3] 対比地 恭平, 天笠 俊之, GPU を用いた高次元データに対する逆 k 最近傍検索の高速化, 信学技報, vol.121, no.314, DE2021-18, pp.19-24, 2021 年 12 月.

[4] CUDA, `https://developer.nvidia.com/cuda-toolkit`.

[5] I. Stanoi, D. Agrawal, and A. E. Abbadi, Reverse nearest neighbor queries for dynamic databases, ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pp.44-53, 2000.

[6] W. Wu, F. Yang, C. Y. Chan, and K.-L. Tan, Finch: Evaluating reverse k-nearest-neighbor queries on location data, Proc. PVLDB Endow., vol.1, no.1, pp.1056-1067, Aug 2008.

[7] S. Yang, M. A. Cheema, X. Lin, and Y. Zhang, Slice: Reviving regions-based pruning for reverse k nearest neighbors queries, IEEE 30th ICDE, pp.760-771, 2014.

[8] Y. Tao, D. Papadias, and X.Lian, Reverse knn search in arbitrary dimensionality, PVLDB, pp.744-755, 2004.

[9] Y. Tao, D. Papadias, X.Lian, and X.Xiao, Multidimensional reverse knn search, VLDB J., vol.16, no.3, pp.293-316, July 2007.

[10] A. Singh, H. Ferhatosmanoglu, and A. Tosun, High Dimensional Reverse Nearest Neighbor Queries, Proc. 12th Int. Conf. Inf. Knowl. Manage., pp.91-98, 2003.

[11] V. Garcia, É. Debreuve, F. Nielsen and M. Barlaud, K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching, 2010 IEEE International Conference on Image Processing, pp.3757-3760, 2010.

[12] cuBLAS, `https://docs.nvidia.com/cuda/cublas/index.html`

[13] H. Jégou, M. Douze, and C. Schmid, Product quantization for nearest neighbor search, IEEE Trans. PAMI, vol.33, no.1, pp.117–128, Jan. 2011.