GPU を用いた Canopy クラスタリングの高速化

林　史尊† 小澤　佑介† 天笠　俊之†† 北川　博之††

† 筑波大学大学院システム情報工学研究科　〒 305–8573 茨城県つくば市天王台一丁目 1 番 1 号
†† 筑波大学システム情報系　〒 305–8573 茨城県つくば市天王台一丁目 1 番 1 号
E-mail: †{hayashi,kyusuke}@kde.cs.tsukuba.ac.jp, ††{amagasa,kitagawa}@cs.tsukuba.ac.jp

あらまし　大規模データに対するクラスタリングは計算コストが高く，高速化が求められている．高速化を実現する手法の一つに Canopy クラスタリングがある．Canopy クラスタリングは，凝集法や k-means 法といった一般的なクラスタリングアルゴリズムを高速に実行するための事前処理として行われる．また，クラスタリングアルゴリズムの並列計算による高速化もさかんに研究されており，その中で並列計算に優れた GPU の利用が注目されている．一般的なアルゴリズムは GPU を用いた高速化が研究されているが，Canopy クラスタリングを GPU 上で行う試みは現在までなされていない．本研究では，GPU を用いて Canopy クラスタリングを実行し，高速化の程度を検証した．

キーワード　データクラスタリング, Canopy クラスタリング, GPU

1. はじめに

近年，科学分野で扱われるデータは非常に膨大なものとなっている．それに合わせて，大規模データの解析に要するコストは肥大化しており，効率的にデータを処理する手法が求められている．データ・クラスタリング（以降，単に「クラスタリング」と呼ぶ）はデータ解析によく用いられる手法で，データ集合をデータ間の類似度に基づきグループ化する処理である．クラスタリング手法は多く提案されており，データの特徴やグループ化の目的によって用いる手法を選ぶ．しかし，クラスタリングは一般に計算コストが高いため，大規模データに適用するのは難しく，計算の高速化が求められている．

クラスタリングの計算高速化方法の一つとして，Canopy クラスタリング [1] が注目を集めている．Canopy クラスタリングは，凝集法や k-means 法といった，よく知られるクラスタリング手法の前準備として行われる．Canopy と呼ばれるおおまかなクラスタを，簡単な計算によって事前に生成しておくことで，後に用いるクラスタリングアルゴリズムでの計算を軽減できる．Canopy の生成は，Canopy の中心点を選んで，その中心から一定の距離内にあるデータを Canopy として登録し，また異なる一定の距離内にあるデータを中心点の候補から取り除くことを繰り返して実現される．生成された Canopy によって距離の近い（類似度の大きい）データを求めているため，その後厳密なクラスタリング計算を行う段階では，すべてのデータ間の距離を求める必要がなくなる．

また，異なるクラスタリングの高速化方法として並列化が挙げられる．これは，データ集合を複数に分割し，各部分集合で処理を同時実行した後，結果を結合させるというアイディアに基づいている．例えば，クラスタリングの各データ間の距離を計算する処理は，並列的に行うことで計算時間を削減できる．ただし，データ分割の方法によって並列処理の効率は大きく変わるため，適切な方法でデータ分割を行うことが重要となる．

一方，計算機処理を並列化する手段の一つとして，GPU（Graphics Processing Unit）コンピューティングがさかんになっている [2]．GPU はグラフィックス処理に特化した演算装置である．小規模なプロセッサを大量に持つという構造の性質上，単純な計算の並列処理に優れる．GPU コンピューティングは，GPU を利用して，グラフィックス処理以外の計算を行うものである．ただし，GPU の構造は特殊であるため，利用の際にはアルゴリズムを最適化する必要がある．

GPU コンピューティングにおける凝集法や k-means 法などについてはすでに研究されているが，Canopy クラスタリングを GPU 上で行う手法は現在まで提案されていない．本研究では，GPU を利用して，Canopy クラスタリング並列的に処理する．GPU 上では，データ間の距離計算などを並列に行って，順次 Canopy を生成する．さらに，ナイーブな Canopy クラスタリングを GPU 上で実行するだけでなく，より高効率に並列処理を行えるよう，セル構造を用いた独自の手法を提案する．独自手法では，データ空間をいくつかのセルに分割し，それを利用して Canopy 生成時の計算範囲を削減することを狙いとする．各データが所属するセルとそれに隣接するセルを求めることで，各データにおける距離計算の対象データは所属・隣接セル内のもののみに限定される．これにより，特に大規模なデータセットに対するクラスタリング計算を高速化できる．独自手法についても GPU 上での処理を行った．セル構造を用いることで 1 個のデータに対応する計算範囲が制限されたことにより，並列して複数の Canopy を生成することが可能となる．

また，本研究では，ナイーブな Canopy クラスタリング，セル構造を用いた Canopy クラスタリングそれぞれについて，CPU および GPU 上で実行した際の計算時間を比較する実験を行った．実験により，2 次元 1,000,000 データを対象にした場合，GPU 上の独自手法が CPU 上のナイーブな手法に比べておよそ 15 倍高速化しているという結果が得られた．

2. 基本的事項

2.1 クラスタリング

クラスタリングとは，データ集合をデータ間の類似性に基づきグループ（クラスタ）化する処理であり，規則性の発見やデータの分類に利用される．具体的には，顔認識，マーケティングリサーチ，文書のトピック別分類などといった，非常に幅広い分野で用いられている．クラスタリングは，その結果が階層構造を持つか否かで，階層型と非階層型の二種類に分けられる．

階層型クラスタリングでは，逐次的にクラスタの結合または分割を繰り返すことで，階層的クラスタ構造を得る．凝集法が代表的手法であり，デンドログラムと呼ばれる階層構造が得られる．凝集法は，各クラスタ間の類似度を計算し，類似度が最も高い 2 個のクラスタを結合することを，クラスタ数が 1 個になるまで繰り返す（終了条件のクラスタ数を設定することも可能）．凝集法の計算量は，データ数を n とすると $O(n^3)$ と大きい．

非階層型クラスタリングでは，クラスタの分割精度を評価する関数を用意し，評価値が最適となる分割を探索する．代表的な手法に k-means 法がある．k-means 法は，重心をいくつか定め，各重心について近いデータを求め，各重心を再計算するという処理を，すべての重心の位置が変化しなくなるまで繰り返す．k-means 法の計算量は，データ数を n，求めるクラスタ数を k，反復回数を t とすると $O(tkn)$ となる．

これらの手法を単純に処理するのは非効率的である．そのため，アルゴリズムの改良や事前処理などが行われる．前者の例としては，F. Murtagh が最近傍グラフを利用し凝集法の計算量を $O(n^2)$ に抑えている [3]．後者の一例には Canopy クラスタリング [1] がある．これは，あらかじめ簡単な計算によりおおまかなクラスタを生成しておくことで，後に行う一般的なクラスタリング計算を軽減するという手法である（詳細なアルゴリズムは 3.1 節で述べる）．

2.2 GPU コンピューティング

GPU はグラフィックス処理に特化した演算装置である．一般的演算を行う CPU（Central Processing Unit）は逐次的な処理に優れるのに対し，GPU は簡単な処理の並列実行に優れる．それまで逐次的に行っていた処理を GPU 上で並列的に行うことで，単に CPU のみを使う場合に比べ実行アルゴリズム全体の計算時間を大きく短縮できると期待される．GPU をグラフィックス処理とは異なる用途で利用することが注目されている [2]．このように，GPU を汎用的計算に活用する技術を GPU コンピューティングと呼ぶ．

CPU は大きなプロセッサを数個持つが，GPU は小さなプロセッサを数百個持つ．GPU 内には複数のストリーミングマルチプロセッサ（SM）が存在し，SM はまた複数のストリーミングプロセッサ（SP）から構成される．SM は一つの処理を行う単位となる．

GPU が利用できるメモリもまた，複数の層から構成される．例えば，各 SM が持つオンチップメモリは，対応する SM から高速アクセス可能だが，数十 KB と容量が小さい．デバイスメモリは，CPU 側からアクセス可能で，かつどの SM からもアクセスでき数百 MB〜数 GB と大容量だが，アクセス遅延が大きい（処理実行の単位時間に比べ数十〜数百倍）．GPU での並列処理を行う際は，各メモリの特徴を捉えてデータ分割を行う必要がある．

GPU を用いた処理の基本的な流れは次のようになる．

（1） CPU 上で，GPU を使った計算に必要なデータを準備する（GPU 上のメモリ確保等）

（2） データを GPU に転送する

（3） カーネル（GPU 上の処理を定義した関数）を呼び出す

（4） GPU による処理

（5） 処理後のデータや結果を必要に応じて CPU に転送する

なお，基本的にカーネル内で動的なメモリ確保は行えない．そのため，どれほどのサイズが必要か分からないデータが存在する場合であっても，カーネル実行前に適当なサイズ分のメモリを確保しなければならない．また，CPU–GPU 間のデータのやり取りは時間がかかるため，データ転送およびカーネル呼び出しの回数はなるべく抑える必要がある．

GPU コンピューティング向けの統合開発環境の一つに，NVIDIA が提供する CUDA（Compute Unified Device Architecture）がある．CUDA は，GPU が行う処理の指示，主記憶と GPU 用メモリ間のデータ転送などをサポートする．プログラミングレベルでは，基本的に C/C++言語のライブラリとして利用する．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（スレッディングモデル）を基軸として行われる．スレッディングモデルは，スレッド，ブロック（スレッドブロック），グリッド（カーネルグリッド）からなる．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いう．グリッドは GPU 上で行う一連の処理に当たる．通常，1 ブロックでは数百個のスレッドを実行し，1 グリッド内では数百〜数千万のブロックが実行される (注1)．実際に使用するスレッド数とブロック数は，処理内容に基づき適切な値をあらかじめ設定しておく必要がある．

論理的なメモリ構造としては，例えば，ブロック処理中にアクセスできるシェアードメモリや，CPU 側からもアクセス可能なグローバルメモリなどがある．それぞれ，シェアードメモリはオンチップメモリに，グローバルメモリはデバイスメモリに対応する．CPU から転送されたデータは，基本的にグローバルメモリに置かれる．また，先述の通り，グローバルメモリ（デバイスメモリ）はアクセス遅延が大きいため，データの再利用などを行う際にはシェアードメモリを活用する．なお，GPU はグローバルメモリやシェアードメモリ以外にも数種類のメモリを内蔵するが，ここでは説明を割愛する．

(注1)：本研究で用いる Fermi アーキテクチャ [5] では，1 ブロック内最大スレッド数は 1024，1 グリッド内最大ブロック数は 65535 × 65535 である．

3. ナイーブな Canopy クラスタリング

3.1 アルゴリズム

Canopy クラスタリングは，McCallum, A. らによって提案された手法であり [1]，凝集法や k-means 法といったクラスタリング手法の前処理として利用される．これまで大きな計算時間を要していた大規模データクラスタリングを，実用的なレベルまで高速化できるため，近年注目されている．例えば，Zhi-Gang Fan らは，画像データのクラスタリングにおいて，Canopy クラスタリングを用いた高速化を図った [6] ．また，Tuli De らは， 700000×1500 という大きなサイズの銀河系外スペクトルデータに対し，Canopy クラスタリングを利用してクラスタリング行った [7] ．

この手法を用いたクラスタリングでは，まず，簡単な距離計算を行ってデータ集合を Canopy と呼ばれるグループに分割する（1 個のデータが複数の Canopy に属してもよい）．その後，凝集法や k-means 法のような厳密なクラスタリングを行うが，ここでの各データ間の距離計算はそれぞれの Canopy の中のみで行われる．クラスタとなり得るデータをあらかじめ Canopy としてまとめておくことで，実際のクラスタリング処理における無駄な距離計算を省ける．

具体的な例を挙げると，基本的な凝集法では，全てのクラスタ組合せについて距離計算を行い，最も距離が小さいものを新たなクラスタとして結合するという処理を繰り返す．特に初めは，各々のデータが 1 個のクラスタとして扱われるため，距離計算が必要な組合せ数は非常に大きい．10000 個のデータについて凝集法クラスタリングを行う場合であれば，1 個のデータにつき，他 9999 個のデータとの距離を求めなければならない．ここで，Canopy クラスタリングをあらかじめ行った場合を考える．あるデータが所属する Canopy が，仮に 1000 個のデータを持っているとする．この状態で凝集法を開始したとき，そのデータと距離計算を行うデータは，同じ Canopy に存在する他 999 個のデータのみでよい．Canopy 外のデータは，元々同じクラスタにはなり得ない（なりにくい）ものとして考えるためである．このように，最終的に同じクラスタを形成しないデータとの距離計算を行わずに済むため，計算回数を抑え処理時間を削減できる．

Canopy を生成するアルゴリズムの流れは以下のようになる（図 1 を参照）．

（1） クラスタリングを行いたいデータセットを用意し，各データのインデックスを中心点候補とする．

（2） 中心点候補の中から 1 個のデータをランダムに選択する．

（3） 2 で選んだデータと他のすべてのデータについて，各々の距離を計算する．

（4） 距離が T_1 以下のデータを Canopy に含める．

（5） 距離が $T_2 (< T_1)$ 以下のデータのインデックスを中心点候補から除外する．

（6） 中心点候補が 0 個になるまで，2〜5 の処理を繰り返す

生成された Canopy は，以下の性質を持つ．

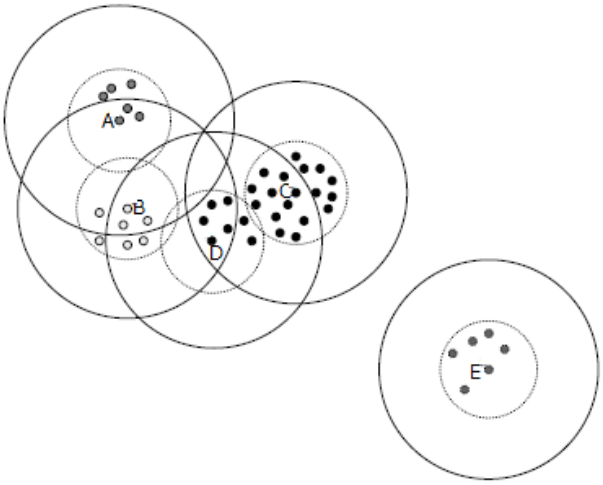

図 1 Canopy の生成．アルファベットのついた点は各 Canopy の中心．実線は中心点からの距離 T_1 を示す．破線は中心点からの距離 T_2 を示す．この例では，まず初めに，すべての点からランダムに 1 個を選択し，選ばれた点（A）を 1 個目の Canopy の中心としている．その後，距離 T_1 内の点を Canopy に含め，距離 T_2 内の点を中心点候補から除外する．これで 1 個の Canopy が生成される．2 個目の Canopy 中心は，残っている中心点候補から新たに選出される．選ばれた点（B）を中心に，A の時と同じように Canopy を生成する．同様の手順で， C，D，E をそれぞれ中心とする Canopy が生成される．E の Canopy が生成された時点で，すべての点が中心点候補から除外された，すなわち各点がいずれか 1 個以上の T_2 円内に含まれたため，Canopy クラスタリングの処理を完了する．（図は”Efficient Clustering of High Dimensional Data Sets with Application to Reference Matching” [1] Figure 1 を引用）

- 複数の Canopy に，同じデータが所属し得る．
- すべての Canopy の中心点について，互いの距離は T_2 より大きい．

なお， T_1 および T_2 の値は，データの性質や距離尺度等をもとに設定する．T_1 は各 Canopy が含むデータ数に関わり， T_2 は Canopy 生成数に関わる．生成した Canopy の大きさや数は，この後に行われる本格的なクラスタリングの計算結果の妥当性，および計算時間に影響する（Canopy クラスタリング後の処理については，本研究では扱わない）．

4. GPU 上での並列処理

GPU を用いてナイーブな Canopy クラスタリングを実現する場合，データセットの準備および終了条件判定を除く以下の処理を GPU 上で行うことができる．

（2） 中心点候補の中から 1 個のデータをランダムに選択する．

（3） 2 で選んだデータと他のすべてのデータについて，各々の距離を計算する．

（4） 距離が T_1 以下のデータを Canopy に含める．

（5） 距離が $T_2 (< T_1)$ 以下のデータのインデックスを中心点候補から除外する．

ここでは主に，3, 4, 5 について説明する．なお，Canopy クラスタリングは他の一般的なクラスタリング手法の事前処理として行われるため，後の処理を GPU 上で行うことを前提として実装を行っている．具体的には，各 Canopy にどのデータが所属しているかという情報を GPU 上に残すようにした．

4.1 中心点の選択と距離計算

処理の簡単のため，新しく選択する Canopy の中心は，常に中心点候補の先頭要素とする．アルゴリズムの性質上，どのデータが中心として選択されたとしても，T_2 近傍にあるデータは必ず中心点候補から削除される．そのため，中心点選択をランダムに行わないことの影響は小さいと考えられる．

続いて，中心点となったデータと各データとの距離計算を，各スレッド上で並列に行う．1 個のスレッドで中心点データと比較対象のデータ 1 個を読込み，その距離を算出する．さらに，同一スレッド上では，算出距離と T_1 および T_2 値との比較を行い，T_1 より距離が小さい場合は T_1 フラグとして 1 を，T_2 より距離が小さい場合は T_2 フラグとして 1 を，それぞれ T_1 フラグ列，T_2 フラグ列に記録する（1 でない場合は 0 を記録する）．

距離計算および T_1, T_2 フラグを求める処理について，簡単な図例を用いて詳しく説明する．図 2 は，8 個のデータについて処理を行おうとするものである．ここで，$T_1 = 7, T_2 = 4$ と設定し，中心点には data3 が選択されたものとする．各スレッドは，まず，スレッド番号に対応するインデックスのデータ（data0〜data7）と，中心点となるデータ（center，この例では data3）を読込む．その後，距離計算を行ったところ，それぞれのスレッドで 3 段目のような結果が得られた．各スレッドは自身の中で求められた距離と T_1, T_2 の値を比較して，割当てられたデータが中心点に対しどのような位置にあるかを，T_1 フラグ，T_2 フラグとして求める．例えば data0 ならば，中心点から距離が 2 であり，これは T_1 および T_2 以下であるため，それぞれのフラグに 1 が記録される．

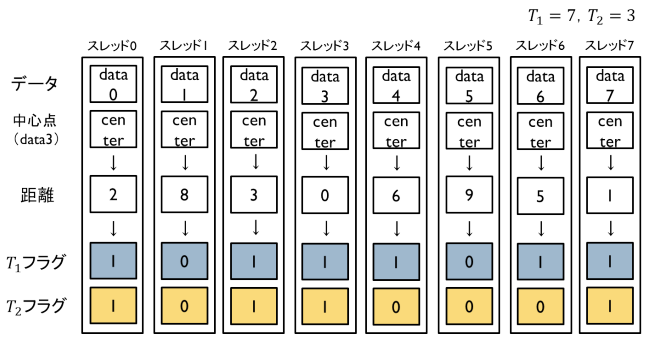

図 2　距離計算の並列化．各スレッドは，スレッド番号に対応するインデックスのデータと，中心点となるデータを読込む（この例では，data3 が中心点として選ばれている）．中心点データと各データで距離計算を行い，その結果によって，T_1, T_2 フラグの値を決定する．

4.2 Canopy の更新

距離計算処理によって得られた T_1 フラグ列を用いて，選択した中心点に対応する Canopy の所属データ（Canopy メンバ）を記録する．このとき，Canopy メンバを更新する場合に必要な情報は，計算対象とする Canopy にどのインデックスのデータが所属しているかである．これは，T_1 フラグ列中で 1 の値を持つインデックスを集めることで得られる．フラグの立ったインデックスを抽出するには，Inclusive Scan [9] と呼ばれる処理を行う．Inclusive Scan では，配列 $Q_{in} = [q_0, q_1, q_2, ..., q_n]$ から配列 $Q_{out} = [q_0, q_0 + q_1, q_0 + q_1 + q_2, ..., q_0 + q_1 + q_2 + ... + q_n]$ が得られる．

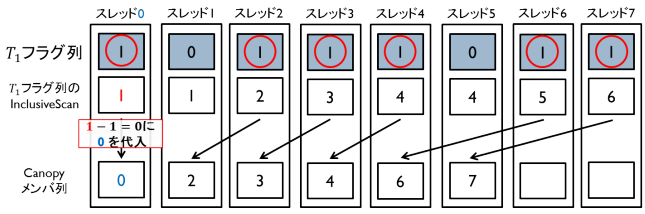

図 3　Canopy の更新．T_1 フラグ列（1 段目）と Inclusive Scan 後の T_1 フラグ列（2 段目）を用意する．T_1 フラグ列中で値が 1 であるインデックスについて，Inclusive Scan 後の T_1 フラグ列の値を確認する．インデックス値を，Canopy メンバ列中の確認した値の位置に格納する．なお，2 回目の Canopy 生成処理以降は，既存の Canopy メンバ数 + Inclusive Scan 後の T_1 フラグ列の値の位置にインデックス値を格納すればよい．

Inclusive Scan 後の T_1 フラグ列は，T_1 フラグが立っているデータの数を先頭から順に記録しているものとみなせる．この特徴を利用することで，T_1 フラグ列と Inclusive Scan 後の T_1 フラグ列から，T_1 フラグの立ったインデックスを抽出した配列が得られる．まず，T_1 フラグ列において値が 1 であるインデックスの，Inclusive Scan 後の値を得る．抽出配列では，この値の位置にインデックス値が格納される（図 3 参照．ただし，配列のインデックスは 0 から数えるため，実際に格納される位置は，Inclusive Scan 後の値 -1 となる）．この抽出配列は計算対象である Canopy のメンバであるため，これを Canopy メンバ列に追加する．

4.3 中心点候補の更新

続いて，T_2 フラグ列を用いて，中心点候補の除去を行う．その時点で残っている中心点候補それぞれの T_2 フラグ列を調べ，値が 1 であるものを除去，あるいは値が 0 であるものを抽出すればよい．これは，以下に示す 3 段階の処理によって実現される．

（1） 中心点候補に残っているデータのインデックスについてのみ，T_2 フラグ列から値を抽出する．また，3 の処理のため，抽出時にフラグの値（ 0,1 ）を反転させる．

（2） 抽出 T_2 フラグ列の Inclusive Scan を計算する．

（3） その時点での中心点候補，抽出 T_2 フラグ列，Inclusive Scan 後の抽出 T_2 フラグ列から，中心点候補として残るデータのインデックスを求める．

1 の処理は単純に，中心点候補であるデータのインデックスそれぞれについて T_2 フラグ列の値を確認し，抽出 T_2 フラグ列に格納する（図 4 参照）．このとき，3 の処理で T_2 フラグ

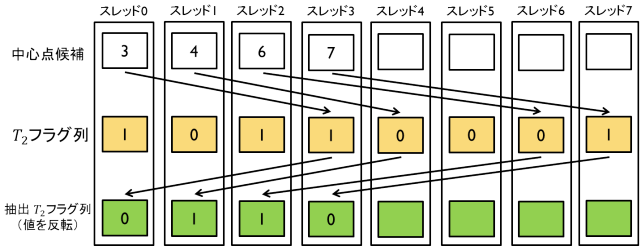

図 4　中心点候補の更新 (1)．更新前の中心点候補（1 段目）それぞれの T_2 フラグの値を，抽出 T_2 フラグ列として得る．更新後は T_2 フラグの値が 0 であるもの（中心点から距離 T_2 以内に存在しないデータ）を残したいため，後の処理に向けて T_2 フラグの値を反転させておく．

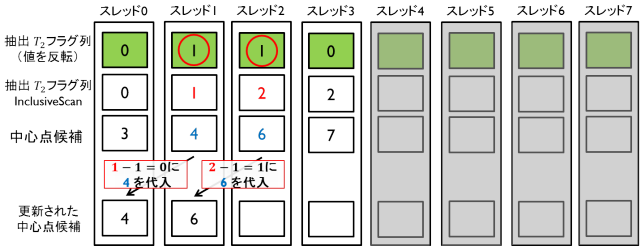

図 5　中心点候補の更新 (2)．抽出 T_2 フラグ列と Inclusive Scan 後の抽出 T_2 フラグ列から，中心点候補が更新後，どの位置に格納されるかが分かる．ここから，T_2 フラグが 1 になっている中心点候補を抜き出せる．

が 0 であるものを抽出するため，T_2 フラグの値を反転させる．

3 の処理の基本的な流れは，Canopy 更新時の処理と同様である．抽出 T_2 フラグ列中で値が 1 であるインデックスについて，Inclusive Scan 後の抽出 T_2 フラグ列の値を確認する．中心点候補の同インデックスに格納されている値を，新しい配列の確認した値の位置に格納し，これを更新後の中心点候補とする（図 4 参照）．なお，抽出 T_2 フラグ列の末尾の値は，更新後の中心点候補の総数を意味する．この情報は CPU 側に転送され，Canopy クラスタリングの終了条件判定に用いられる（値が 0 であれば Canopy クラスタリングを終了する）．

5.　セル構造を用いた Canopy クラスタリング

5.1　基本アイディアとアルゴリズム

本研究で提案する独自の手法は，複数 Canopy の並列生成を狙いとする．4 節では，ナイーブな Canopy クラスタリングのアルゴリズムと，GPU による並列化の方法について述べた．この方法は，1 個の Canopy 中心点について，データセット中の全データとの距離計算を行い，Canopy メンバと中心点候補の更新を行うという処理を繰り返すものだった．しかしながら，1 個の Canopy に含まれるデータは，中心点から距離 T_1 以内に存在するもののみである．中心点から明らかに離れているデータについても計算を行うのは非効率的である．中心点近傍に存在するデータだけを計算対象にすると，処理に必要な計算回数が減少する．こうして Canopy 1 個あたりの計算範囲を制限することで，複数の Canopy を並列に生成できると考えられる．

計算範囲を決定するために，独自手法ではセル構造を用い

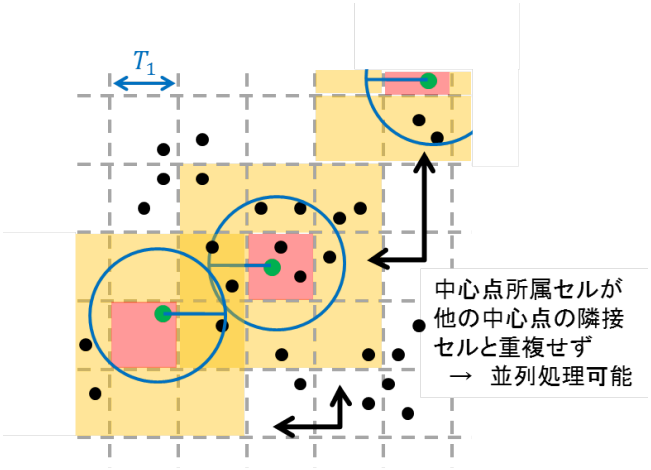

図 6　セル構造を用いた複数 Canopy の並列生成．この例は，2 次元データ空間にセルを生成したものである．中心点（緑）を，計算範囲（赤および橙のセル）が重複しないように選択する．これにより，複数の Canopy を並列して生成することが可能となる．

る．データ空間を間隔 T_1 のグリッド状に分割し，分割された個々の領域をセルと呼ぶ．このとき，すべてのデータは必ずいずれか 1 個のセルに含まれる．ここで，ある 1 個のデータが Canopy の中心点に選ばれたとする．この中心点に対する計算範囲は，中心点が所属するセルと，それに隣接するセルになる．セルのサイズから，その Canopy に含まれる（中心点から距離 T_1 以内にある）データは，中心点の所属セルおよび隣接セル内にしか存在し得ないためである．同様に，中心点と計算範囲が重複しないように，異なる中心点を可能な限り選択する（図 6 参照）．選択後，すべての計算範囲内のデータについて，距離計算，Canopy および中心点候補の更新を行う．

独自手法のアルゴリズムは以下のようになる．

（1）　クラスタリングを行いたいデータセットを用意し，各データのインデックスを中心点候補とする．

（2）　各データの所属セルを計算し，それぞれの所属セルに対応する隣接セルを求める．(各所属セルはセル番号を与えて管理する)．具体的には，全データについて以下の処理を行う．

(a)　各次元の値と T_1 との商（小数点以下切り捨て）を求め，所属セルの位置情報とする．

(b)　同じ位置情報を持つセルがそれまでに現われていなければ，その所属セルに新しいセル番号（直前の番号 +1）を与える．

(c)　新しいセル番号が現われたとき，これまでに現れたセルの位置情報と比較して，すべての次元について差が 1 以下であるセルを，その所属セルの隣接セルとする．

（3）　中心点候補の中から，計算範囲が重複しないように複数の中心点を選択する．具体的には，以下の処理を行う．

(a)　この時点での中心点候補と同一の配列を用意し，これを cp 中心点候補とする．

(b)　選択した中心点を格納する配列を用意し，これを中心点列とする．

(c)　cp 中心点候補の先頭を選択して取り出し，中心点列

に格納する．

(d)　選択した中心点の所属セル，またその隣接セルに所属するデータを，cp 中心点候補から除去する．

(e)　cp 中心点候補が 0 個になるまで，c，d の処理を繰り返す．

（4）　各中心点に対応する計算範囲内のデータを抽出する．具体的には，以下の処理を行う．

(a)　各中心点に対応する計算範囲内のデータのインデックスを格納する配列を用意し，これを各中心点対応データメンバ列とする．

(b)　各中心点について，その所属セルおよび隣接セルに所属するデータを，各中心点対応データメンバ列に加える．

（5）　3 で選んだ複数の中心点と，それぞれの中心点に対応する計算範囲内のデータについて，各々の距離を計算する．

（6）　距離が T_1 以下のデータを Canopy メンバに追加する．

（7）　距離が $T_2(< T_1)$ 以下のデータのインデックスを中心点候補から除外する．

（8）　中心点候補が 0 個になるまで，3〜7 の処理を繰り返す

ナイーブな Canopy クラスタリングと比較して，2〜4 の処理が追加されている．4 の処理後の基本的な流れはナイーブな手法と変わらない．

2 については，処理が逐次的で GPU 上での並列処理に不向きであるため，CPU 上で行うものとする．ここでは，全データについて，各次元の値からそのデータがどのセルに所属しているかを求める．実際には，それぞれのデータに対し，各次元の値とセルのサイズの商（小数点以下切り捨て）をそのデータが所属するセルの座標とする．セルにはセル番号が与えられ，既存しない座標のセルが現われた場合には，そのセルに新しい番号を与える．データとセルは，データ番号およびセル番号を用いて互いの所属情報を持つ．ここで，セルのサイズを一辺 T_1 とすることにより，あるデータが Canopy 中心点として選択されたとき，その Canopy のメンバとなり得るデータは，その中心点の所属セルとその隣接セルのみであると判定できる．Canopy 計算時には，各中心点に対して所属・隣接セル内のデータを求めることで，距離計算の回数を削減することができる．

あるセル $C_n = (c_{n,1}, c_{n,2}, ..., c_{n,d})$ において，その隣接するセルの座標は，C_n との各次元の差が ± 1 であるという条件を満たす．これを利用することで，隣接セルを以下のように求められる．

（1）　既存セルを，小さい次元を優先して昇順ソートする（辞書順ソート）．

（2）　k の初期値を 0 として，以下の処理を再帰的に繰り返す．

（a）　$c_{n,k}$ と各既存セルの k 次元目の座標を比較する．

（b）　初めて $c_{n,k} - 1 < c_{i_{-1},k}$ となる位置 i_{-1}，初めて $c_{n,k} > c_{j_{-1},k}$ となる位置 j_{-1} を求める．

（c）　初めて $c_{n,k} - 1 < c_{i_{+1},k}$ となる位置 i_{+1}，初めて $c_{n,k} > c_{j_{+1},k}$ となる位置 j_{+1} を求める．

（d）　$[i_{-1}, j_{-1}], (j_{-1}, i_{+1}), [i_{+1}, j_{+1}]$ の範囲内それぞれで，$k+1$ 次元目の座標を比較する．$k = d$ のとき，範囲内にある

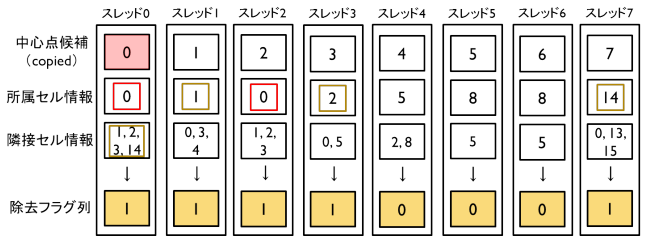

図 7　複数中心点の選択．1 段目は cp 中心点候補，2 段目は各候補の所属セル番号，3 段目は所属セルの隣接セルのセル番号である．cp 中心点候補には所属・隣接セルに所属しないデータを残すため，これらに残存フラグを立てる．その後は，3.2.3 節の処理と同様に進め，cp 中心点候補を更新する．

$c_{n,d} \pm 1$ と等しい座標を持つセルを，C_n の隣接データとする．

5.2　GPU 上での並列処理

ここでは，5.1 節で示したアルゴリズム中の，主として 3 と 4 の処理について詳しく説明する．なお，2 の処理は CPU 上で行うものとする．これは，2 の処理が逐次的であり，並列化に適していないためである．

5.2.1　複数中心点の選択

中心点を，その計算範囲（所属セル＋隣接セル）が重複しないように複数選択する．そのためには，新しく中心点を選択する度に，所属・隣接セル内のデータを候補から除去していけばよい．GPU では，以下のような処理を行う．

(a)　この時点での中心点候補と同一の配列を用意し，これを cp(copied) 中心点候補とする．

(b)　選択した中心点を格納する配列を用意し，これを中心点列とする．

(c)　cp 中心点候補の先頭を選択して取り出し，中心点列に格納する．

(d)　選択した中心点の所属セル，またその隣接セルに所属しないデータについて残存フラグを立てる（図 7 参照）．

(e)　残存フラグ列の Inclusive Scan を計算する．

(f)　cp 中心点候補，残存フラグ列，Inclusive Scan 後の残存フラグ列を利用して，4.2 節で説明した処理と同様に cp 中心点候補を更新する．

(g)　cp 中心点候補が 0 個になるまで，(c)〜(f) の処理を繰り返す．

5.2.2　各中心点に対応する計算範囲内のデータ列抽出

この処理は，複数中心点の選択と同一のカーネルで行うことができる．各中心点に対応するデータメンバ列として抽出するデータは，中心点の所属セルとその隣接セル中にあるため，計算内容は複数中心点の選択とほぼ変わらない．

複数中心点の選択時，選択した中心点に対応する計算範囲内のデータは，候補から除去するデータと同一であるため，これを利用して各中心点に対応する計算範囲内のデータを抽出することができる（図 8 参照）．また，Inclusive Scan 後の残存フラグ列の末尾の値は，加えた対応データメンバ列の総数と一致するため，これを対応データ境界列に格納しておく．

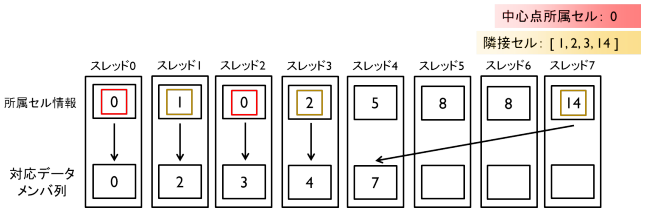

図 8　各中心点に対応する計算範囲内のデータ列抽出．この例では，注目する中心点の所属セルのセル番号を 0，その隣接セルのセル番号を 1，2，3，14 とした場合を考える．これらのセルに含まれるデータは，複数中心点の選択時にその候補から除去するデータと同一であるため，これをそのとき注目している中心点に対応するデータメンバとして得る．その後，別の中心点に対して同じ処理を繰り返す場合は，新しく得られるデータ列を対応データメンバ列の末尾に加えていく．

5.2.3　Canopy の生成と中心点候補の更新

Canopy の生成の際は，1 個の中心点に対する距離計算を 1 個のブロックで行い Canopy メンバを決定する．また，各ブロックで中心点から T_2 以下の距離であるデータをグローバルメモリ中の T_2 フラグ列に記録しておく．すべてのブロックで Canopy メンバを求めた後で，T_2 フラグ列を基に中心点候補の更新を行う．

6.　評 価 実 験

ここでは，ナイーブな Canopy クラスタリング，提案手法であるセル構造を用いた Canopy クラスタリングについて，CPU および GPU 上で行った計算速度の比較実験の結果と考察を示す．

6.1　実 験 環 境

今回の実験で用いるマシンの性能を以下に示す．

- CPU
  - Intel ®Xeon(R) E5630 (2.53GHz * 4 core)
  - OS : Ubuntu 12.04 LTS (64bit)
  - メモリ : 4.0GB
- GPU
  - NVIDIA GeForce GTX 460 (1350MHz * 336 core)
  - メモリ : 767MB

実験に用いたデータセットは人工のものであり，2〜6 次元のベクトルの集合である．データ数は 100〜1000000 個まで変化させる．このデータセットでは，$(\text{データ数} \times 10)^2$ の大きさのデータ空間中に，正規分布を利用して各データをある程度偏在するように配置している（一例を図 9 に示す）．

今回，データ間の距離尺度としてユークリッド距離を採用した．T_1, T_2 の値は実験ごとに固定とする．また，$T_2 = 0.7T_1$ とする．ただし，これはあくまで本実験のために定めたものであり，実際の T_1, T_2 値は，データの性質や求める結果などに応じて定められるべきである．

計算時間の測定については，Canopy クラスタリングの処理部分のみを対象とする．CPU–GPU 間のデータ転送時間もこれに含めるものとする．

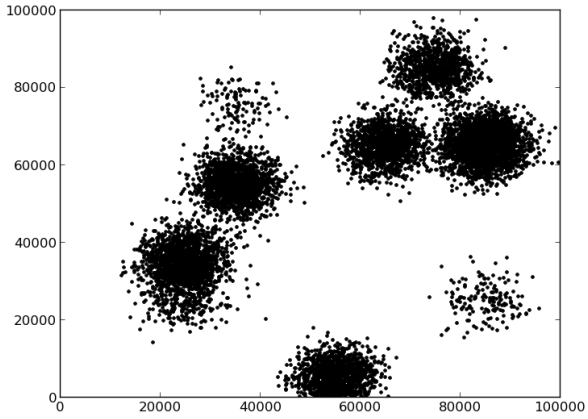

図 9　実験に用いたデータセットの例．100000 × 100000 の大きさを持つ 2 次元空間に，10000 個の点を生成したもの．正規分布を利用し，クラスタとなるようなむらを作っている．

6.2　実 験 結 果

以下の 4 手法について，データ数を固定し次元数を変化させる場合と，次元数を固定しデータ数を変化させる場合の 2 通りについて，計算時間を比較する実験を行った．

- CPU 上でのナイーブな Canopy クラスタリング
- GPU 上でのナイーブな Canopy クラスタリング
- CPU 上でのセル構造を用いた Canopy クラスタリング
- GPU 上でのセル構造を用いた Canopy クラスタリング

以下に，それぞれの場合における実験結果を示す．

6.2.1　実験 1：次元数の変化による計算時間の変化

データ数を 100000 個に固定し，次元数を 2〜6 に変化させて，4 手法の計算時間を比較した．また，$T_1 = 100000, T_2 = 70000$ に固定した．結果のグラフを図 10 に示す．

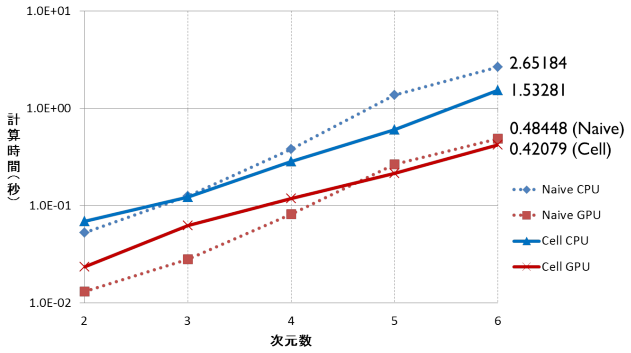

図 10　実験結果（次元数を変化）

いずれの手法においても，次元数に比例して計算時間が増加している．CPU 同士で比較すると，次元数が大きい場合，セル構造を用いた手法が計算高速化に貢献していることが分かる．次元数が大きくなるほどセルの数が増加し，各々の中心点に対応する計算範囲内のデータが減少するためであると考えられる．GPU 上の手法では，両者に大きな差は見られなかった．これは，提案手法において，CPU 上で行ったセルを生成する

処理が計算時間に含まれているためである．セル生成処理に費やされる時間は，例えばデータ数 100,000 のとき全体のおよそ 10〜25%ほどであった．また，本研究で実装した GPU 化アルゴリズムは，完全には最適化されていないと考えられ，工夫によってさらなる高速化が見込める．CPU と GPU で比較すると，GPU 上での計算によって大きく高速化できていることが見て取れる．

6.2.2 実験 2：データ数の変化による計算時間の変化

次元数を 2 に固定し，データ数を 100〜1000000 に変化させて，4 手法の計算時間を比較した．また，$T_1 = 0.5 * \text{データ数}$,$T_2 = 0.35 * \text{データ数}$ に固定した．結果のグラフを図 11 に示す．

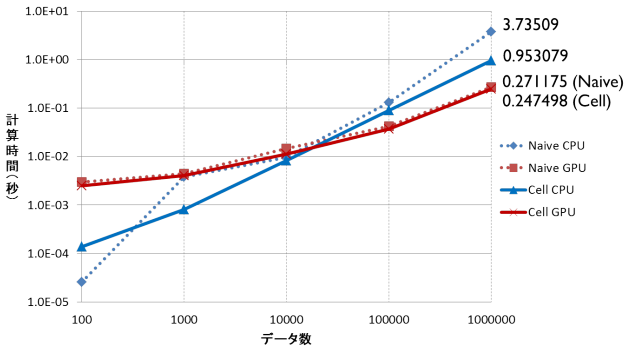

図 11　実験結果（データ数を変化）

データ数が小さいときに GPU 上の手法の計算時間が大きくなっているのは，CPU–GPU 間のデータ転送のオーバヘッドが大きいためである．また，CPU 上の独自手法もデータ数が小さい場合に計算時間がナイーブなものより大きくなっているが，これは事前のセル生成処理による計算回数削減の影響が小さいためである．次元数 2，データ数 1,000,000 のとき，ナイーブな Canopy クラスタリングと GPU 上の独自手法を比較すると，計算時間がおよそ 15 倍速くなっていることが分かる．

7. 結　　論

本研究では，大規模データを対象とするクラスタリングを高速化するため，Canopy クラスタリングの GPU を利用した並列計算を試みた．また，セル構造を用いてさらに並列度を高める独自の手法を提案した．データ数，次元数を変化させ実験を行った結果，独自手法がナイーブな Canopy クラスタリングと比較し，次元数 2，データ数 1,000,000 であるとき，およそ 15 倍高速であることが分かった．

今後解決すべき課題として，GPU のメモリ容量に関する問題がある．GPU のメモリ容量は CPU に比べ小さく，データサイズによってはすべてのデータを GPU に転送できない場合がある．この問題の対策として，複数の GPU を用いる研究 [8] を参考に，CPU 側でデータセットを複数に分割し，分割されたサブデータセットごとに GPU へ転送し処理するという方法が考えられる．

謝辞　本研究の一部は，平成 25 年度共同研究（富士通研究所 CPE25149），JSPS 科研費（24240015），「エクサスケール計算技術開拓による先端学際計算科学教育研究拠点の充実」事業によるものである．

文　　献

[1] A. McCallum, K. Nigam, L. H. Ungar, Efficient Clustering of High Dimensional Data Sets with Application to Reference Matching, Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 169–178 , 2000.

[2] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, J. C. Phillips, GPU Computing, Proceedings of the IEEE, volume 96, Number 5, pp. 879–899, 2008.

[3] F. Murtagh, A Survey of Recent Advances in Hierarchical Clustering Algorithms, The Computer Journal, Vol.26, pp. 354–359, 1983.

[4] NVIDIA, `http://www.nvidia.com`.

[5] NVIDIA, NVIDIA Partnership Training, 2011.

[6] Zhi-Gang Fan, Yadong Wu, Bo Wu, Maximum Normalized Spacing for Efficient Visual Clustering, Proceedings of 19th ACM International Conference on Information and Knowledge Management (CIKM 2010), pp. 409–417, 2010.

[7] Tuli De, Didier Fraix Burnet, Asis Kumar Chattopadhyay, Clustering large number of extragalactic spectra of galaxies and quasars through canopies, Communication in Statistics - Theory and Methods (2013), 2013.

[8] Mohiuddin K. Wasif, P. J. Narayanan, Scalable clustering using multiple GPUs, Proceedings of the 2011 18th International Conference on High Performance Computing, pp.1–10, 2011.

[9] Yan Shengen, Long Guoping, Zhang Yunquan, StreamScan: Fast Scan Algorithms for GPUs Without Global Barrier Synchronization, Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 229–238, 2013.