3K-2

GPUを用いたk-means法の実装と評価

吉村 匠　　小柳 滋
† 立命館大学 情報理工学部

1 はじめに

Graphics Processing Units (GPU) を用いて汎用目的の並列処理を行う手法を General Purpose computing on Graphics Processing Units (GPGPU) と呼ぶ. それは近年広く利用されるようになっており, 例えば東京工業大学のスーパーコンピュータ TSUBAME にも利用されている. GPU は CPU と比べて単純ながらも多数のコアを持っており, 並列性の高い問題に対して価格あたり, および消費電力あたりの性能が優れている. データ並列処理部分を CPU の処理から切り離し, GPU 側で処理することによって, 全体の処理速度を高速化することが可能である. しかし, GPU を用いてプログラミングを行うには, 並列処理やデバイスのメモリ構造などのアーキテクチャに関する専門知識や, 各環境独自のプログラミングモデルの学習など, 実装するためには高度な技術が必要となる. また GPU を導入するにあたっての初期コストや運用コストが問題になることもある.

k-means 法はクラスタリングの中でも頻繁に利用される方法であり、多数あるデータをクラスタが k 個になるまで分割する手法である。本稿では GPU のみで kmeans 法のプログラム処理を実装し、実行時間により評価する。

2 k-means法

k-means 法は, 以下のようなアルゴリズムである. オブジェクトの総数を n, 望むクラスタ数を $k(\leq n)$ とする.

1. k 個のクラスタの重心の初期値を設定する;
2. 直前のクラスタと同一になるまで以下をの手順 3, 4 繰り返す
3. 重心に従ってオブジェクトをクラスタに分けて配置;
4. 配置されたオブジェクトに基づき, 各クラスタの重心を計算する;

上記のステップ 2 から 4 を繰り返すと重心の移動が少なくなってくるので, 上記のアルゴリズムは停止する.

k-means 法の利点は, 単純なアルゴリズムで計算することができることである. そのため, 現在では広く用いられている. しかし, 初期値に大きく依存することが知られており, 1 回の結果で最良のものが得られるとは限らないこと. さらに, 計算量はオブジェクト数とクラスタ数に大きく依存するという欠点がある.

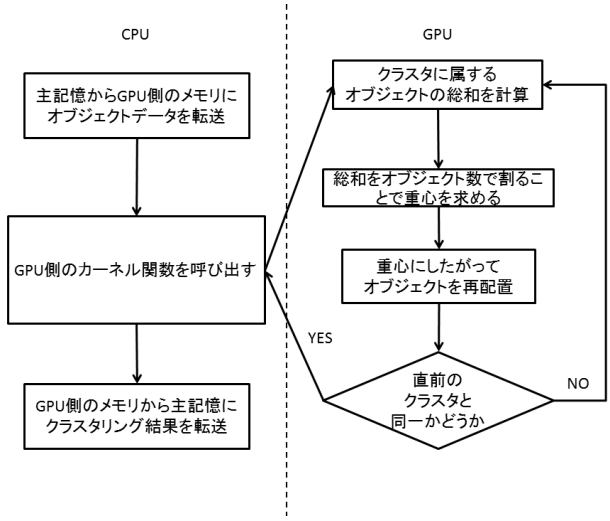

図 1: CUDA を用いた k-means 法

3 実装と評価

3.1 実装

CUDA/GPU で実装する k-means 法の処理の流れを図 1 に示す. まず, オブジェクトをランダムに設定した重心の初期値に従ってクラスタ分けをする. その後, オブジェクトのデータを主記憶から GPU 上のメモリに転送する. GPU 側ではクラスタに属するオブジェクトの総和の計算, クラスタごとの重心計算処理, オブジェクトの再配置処理, 比較処理のカーネル関数を実装する. CPU 側から GPU 側のカーネル関数を呼び出して実行し, オブジェクトの再配置を繰り返して最終的にクラスタ内のオブジェクトの総和の値が変化しなくなるまで繰り返す. 変化しなくなったクラスタのデータを GPU 上のメモリから主記憶に転送して, クラスタリングの完了となる.

各重心との距離の算出処理とクラスタ内のオブジェクトの総和計算処理は, オブジェクト数の増加とクラスタ数の増加につれて計算量も増加する. どちらの処理もメモリアクセス数がとても多くなり, レイテンシも大きくなるため, 初期値と結果の受け渡しのときのみメモリアクセスをすることでレイテンシを軽減する.

Implementation and Evaluation of k-means method on GPU
†Takumi YOSHIMURA　†Shigeru OYANAGI
†College of Information Science and Engineering Ritsumeikan University

表 1: 実行環境

CPU	Intel Core i7 3.4GHz
GPU	NVIDIA GeForce GTX 680
Memory	8GB
OS	Windows 7 Enterprise 64bit

表 2: GPU と CPU の演算結果

オブジェクト数	GPU 実行時間
1024(= 2^{10})	14.05896
4096(= 2^{12})	25.82990
16384(= 2^{14})	58.54855
65536(= 2^{16})	423.34371
262144(= 2^{18})	3130.28857

3.1.1 重心の算出

オブジェクト数が N, クラスタ数が k の場合 $(N \times k)$ 距離行列を確保する. クラスタ毎にオブジェクトの総和により求める. その総和をクラスタ内のオブジェクトの数で割ることで各クラスタ内の平均である重心を求める. オブジェクト毎にクラスタ数だけスレッドを立て, 各オブジェクトと重心との距離を並列に計算する.

3.1.2 オブジェクトの配置

各重心との距離により, オブジェクトがどのクラスタに所属しているかを変更する. 変更する処理は, 全てのオブジェクトを並列に処理することで一度の最も遅いオブジェクトにかかる時間で処理を終えることができる. 全てのオブジェクトに対し変更する処理を施したあと, 全てのオブジェクトの所属クラスタが変更前のクラスタと変化がなければ終了とする.

3.2 評価

CPU と GPU での速度を比較するため, C 言語で k-means 法を実装した. 今回の評価では, 乱数を用いて 1 次元座標を持つオブジェクトを生成したデータセットを使用した. また, プログラムの実行時間の計測に使用した評価環境を表 1 に示す.

図 2 はクラスタ数を固定しオブジェクト数を増加させていった場合の GPU の実行時間の変化を記したグラフである. 詳細は表 2 に示す. 実行時間は 5 回実行した平均値である. 262144(= 2^{18}) 個のデータセットを用いた時, CUDA 実装が 3130.28857(msec) となる.

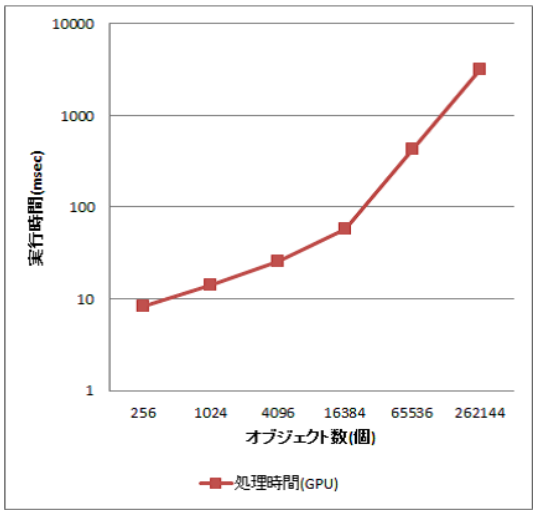

図 2: GPU 処理時間

4 おわりに

本研究では, k-means 法を CUDA/GPU で実装した. 262144 個 (= 2^{18}) のデータセットを用いて実験した結果, 処理全体で 3130.28857(msec) であることが判明した.

今回は GPU のみでの実装となったが, 現段階でも改善点が見られるので更なる高速化を目指す.

参考文献

[1] NVIDIA. https://developer.nvidia.com/category/zone/cuda-zone , 2013

[2] 青木 尊之, 額田 彰: はじめての CUDA プログラミング, 工学社, 2009