5H-01

Sparkにおけるリダクション及びトランスフォーメーション処理のGPU実行

大野 泰弘† 森島 信†† 松谷 宏紀†

†慶應義塾大学 理工学部 情報工学科†
††慶應義塾大学大学院 理工学研究科††

1. 概要

Spark は大規模データ向けの分散処理フレームワークである[1][2]. Spark はデータを処理する際, RDD(resilient distributed dataset)というデータ形式に変換して使用時にそれをメモリに保持するため, ディスクアクセス回数が少なく反復処理や機械学習に向いている[3]. Spark では, RDD に対するリダクション及びトランスフォーメーション処理においてソートなどの計算量の多い処理が多く, ボトルネックとなっている. そこで, 本論文では, Spark におけるリダクション及びトランスフォーメーション処理を, GPU を用いて高速化する.

2. 関連研究

2.1. HeteroSpark

文献[4]では, Spark において CPU に加えて GPU を使用するフレームワークが提案されている. しかし, RDD をどのように GPU で処理しているかについて具体的に言及されていない.

2.2. GPGPU を用いた Spark のグラフ処理の高速化

文献[5]では, Spark のグラフ処理を, GPGPU を用いて高速化することを提案している. その際, RDD を GPU で扱えるデータ構造に変換してから GPU で処理し, 結果を CPU で RDD に変換してグラフ処理を高速化している.

3. Spark RDD メソッドの GPU 実行

3.1. システム全体像

図 1 に Spark から GPU を用いて処理を行う流れを示す. 処理 1 で Spark メソッドが呼ばれたら作成された RDD を, 配列化メソッド(collect)を用いて配列化し GPU に転送して処理を行う. 処理 2 以降では一度作成した配列を再利用することで配列化オーバヘッドを削減する. また, 別の配列を転送するまで GPU には同じ配列を持たせ続けることで転送オーバヘッドも削減する.

3.2. RDD の配列キャッシュ

図 2 に RDD から作成した配列をキャッシュする構造を示す. RDD のデータを GPU で処理する際に作成した配列をキャッシュすることで, 二回目以降の計算では配列を再利用できるようにし, RDD から配列を作成するオーバヘッドを削減する. RDD から配列を作成するとき元の RDD の ID と配列を RDD-配列対応表に格納する. この対応表に格納された, RDD から作成された配列を以後配列

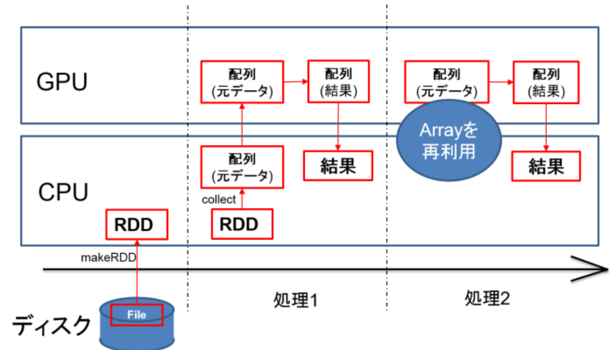

図 1 RDD の配列化と GPU 処理の流れ

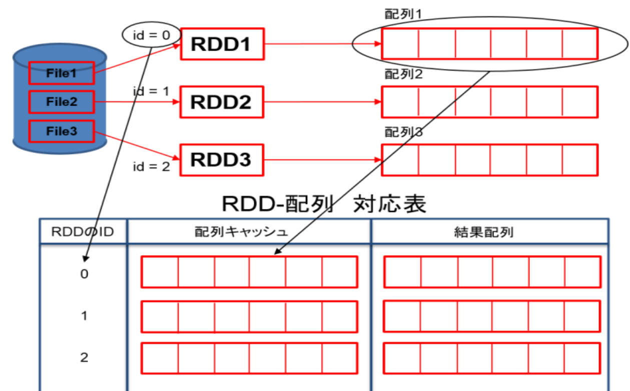

図 2 RDD の配列キャッシュ

キャッシュと呼ぶ. 結果配列は GPU で処理された結果の配列である. RDD から配列を作成して RDD-配列対応表の要素に加えるまでの操作は,

(1) RDD の配列化
(2) RDD の ID と配列を RDD-配列対応表に登録

のみで行える.

3.3. リダクション

図 3 に Spark から GPU を用いるメソッドが呼ばれた際の配列キャッシュを用いたリダクション演算の流れを示す. GPU を用いるメソッドが呼ばれると, RDD-配列対応表中に格納されている要素の ID で, 呼び出し元の RDD の ID と一致するものがあるか調べる. 一致するものがあれば一致した要素の配列キャッシュを GPU に転送し, 一致するものがなければ呼び出し元の RDD の ID とそれを配列化したものを RDD-配列対応表に格納した後配列キャッシュを GPU に転送する. 転送した配列に対し GPU 側では呼び出されたメソッドを実行するためのカーネル関数を実行する.

また, RDD-配列対応表に登録されている ID と呼び出し元の RDD の ID が一致した際, 一致した要素が RDD-配列対応表の何番目か保持しておく. これにより次回メソッドが呼ばれたとき, 前回一致した要素の ID と呼び出し元の RDD の ID が

GPU Execution of Reduction and Transformation Processing on Spark
†Yasuhiro Ohno, Dept. of Information and Computer Science, Keio University
††Shin Morishima, Graduate School of Science and Technology, Keio University
†Hiroki Matsutani, Dept. of Information and Computer Science, Keio University

Copyright ©2016 Information Processing Society of Japan.
All Rights Reserved.

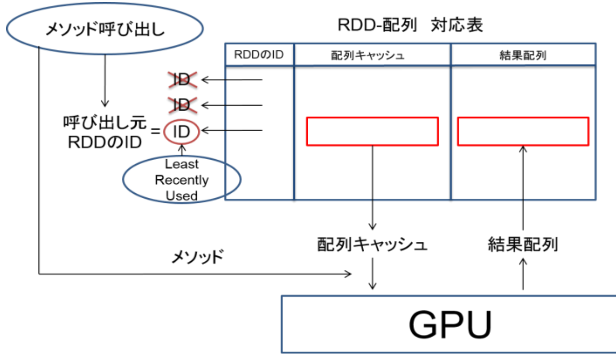

図 3 Spark メソッドから GPU を呼び出す流れ

一致するか調べ，一致した場合 GPU に転送してある配列を再利用できる．この構造により同じ RDD に対し複数回リダクションを行う場合 2 回目以降は GPU への転送オーバヘッドが削減される．

今回 GPU を用いて高速化したリダクションは，要素の総和を求める演算である．要素の総和を求める演算は，Spark における“reduce”メソッドに対応している．

3.4. トランスフォーメーション

トランスフォーメーションの GPU 処理に関しても，前項で述べたリダクション処理と概ね同じである．リダクションと異なるのは，GPU による処理結果を結果配列として保存した後，結果配列をもとにした RDD を作成する点である．

今回 GPU を用いて高速化したトランスフォーメーションはキーソートである．キーソートは Spark における“sortByKey”メソッドに対応している．GPU のカーネル関数で行うソートではバイトニックソートを使用した．

4. 評価

4.1. 評価環境

今回使用した CPU は Intel Xeon E5-1620 v2 で CPU コア数は 4，動作周波数は 3.70GHz でメモリ容量は 128GB である．OS は CentOS6.7 で GPU は GeForce GTX 780 Ti を用いた．CUDA のバージョンは 7.5，jcuda のバージョンは 0.7.5，Spark のバージョンは 1.1.1，scala のバージョンは 2.10.4 を用いた．また，主な GPU の諸元を表 1 に示す．

表 1 GPU の主な諸元

性能項目	GeForce GTX 780 Ti
コア数	2880
コアクロック	875MHz
メモリクロック	7.0Gbps
メモリバス幅	384bit
メモリバンド幅	336GB/s
メモリ容量	3GB

4.2. リダクション

今回のリダクションの性能評価には，3.3 節で述べた要素の総和を求める演算を用いた．評価には約 1GB の INT 型配列を用いた．実行時の java ヒープ領域は 30GB とした．図 4 に Spark と GPU による要素の総和を求める演算を 5 回連続実行した際のそれぞれの内訳を示す．GPU 実行は Spark 実行に対し 1 回目は 6.2 倍の性能向上，2 回目以降は 15.2-24.2 倍の性能向上を達成した．

4.3. トランスフォーメーション

今回のトランスフォーメーションの性能評価

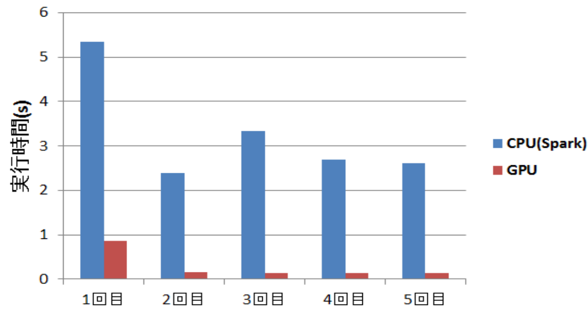

図 4 総和演算の実行時間比較

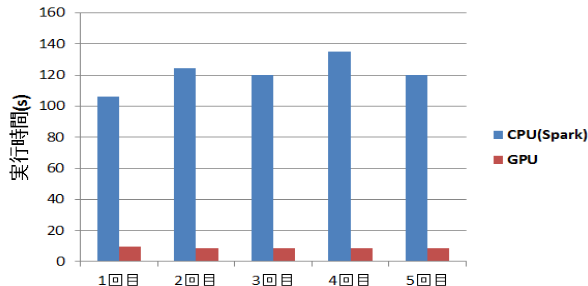

図 5 キーソートの実行時間比較

には，3.4 節で述べたキーソートを用いた．評価には約 1GB の INT 型の(K,V)配列を用いた．実行時の java ヒープ領域は 30GB とした．図 5 に Spark と GPU によるキーソートを 5 回連続実行した際のそれぞれの内訳を示す．GPU 実行は Spark 実行に対し 1 回目は 11.1 倍の性能向上，2 回目以降は 14.2-15.7 倍の性能向上を達成した．

5. まとめ

本論文では Spark RDD メソッドのうち，計算インテンシブな処理を GPU 処理させることで高速化した．また，配列キャッシュを用いて 2 回目以降の処理の配列化オーバヘッドも削減した．

参考文献

[1] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster Computing with Working Sets,” Proceedings of the USENIX Conference on Hot Topics in Cloud Computing (HotCloud’10), pp.10-10, June 2010.

[2] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, Learning Spark Lightning-Fast Big Data Analysis, O’Reilly Media, 2015.

[3] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, and I. Stoica, “Resilient Distributed Datasets: A FaultTolerant Abstraction for In-Memory Cluster Computing,” Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI’12), pp.15-28, April 2012.

[4] P. Li, Y. Luo, N. Zhang, and Y. Cao, “HeteroSpark: A Heterogeneous CPU/GPU Spark Platform for Machine Learning Algorithms,” Proceedings of the International Conference on Networking, Architecture and Storage (NAS’15), pp.347-348, Aug. 2015.

[5] 稲本裕貴，青山幹雄，“Spark の GPGPU を用いたグラフ 処理高速化方法の提案と評価，” Proceedings of the 電子情 報通信学会技術研究報告 SC2014-19, pp.31-36, March 2015.

Copyright ©2016 Information Processing Society of Japan.
All Rights Reserved.