3J-04

GPGPU を用いた Spark によるグラフ並列処理方法の提案と評価

稲本 裕貴† 青山 幹雄‡

南山大学大学院 理工学研究科 ソフトウェア工学専攻† 南山大学 理工学部 ソフトウェア工学科‡

1. 研究背景と課題

グラフ構造のビッグデータを高速処理するための並列分散処理の必要が高まっている. このため Hadoop に対してインメモリ型の Spark[1]が提案された. 本稿では, ハードウェアレベルでの高速化が可能である GPGPU を使用する, Spark の高速処理方法を提案する. このため, 異なる並列計算環境を組み合わせ高速化する方法を提案する.

2. 関連研究

2.1. Spark による高速グラフ処理

Spark の RDD(Resilient Distributed Dataset)をグラフデータに対して拡張を行い, グラフ処理の高速化を実現した GraphX [5]がある. しかし拡張されたデータは GPU での処理に適していない構造のため, GPU を用いた高速化ができない.

2.2. GPU を用いた MapReduce の高速化

Hadoop の MapReduce を GPU で高速化する提案がある[2,6]. しかし, インメモリ型の Spark に対して具体的な提案はされていない.

3. アプローチ

GPGPU を用いて Spark のグラフ処理を高速化する. Device Memory(以下 D_Memory)上で RDD の特徴を活かしつつ GPU での処理を行うために, D_Memory 上のデータを 2 種類に分類する. 実行するグラフ処理は GPU で高速化の効果が最も期待できる, 浮動小数点で結果を求めるアルゴリズムとする.

4. GPGPU を用いた Spark の高速化

GPU で処理を行う Worker を G_Worker とする.

4.1. GPGPU を用いた Spark アーキテクチャ

GPGPU を用いた並列高速化 Spark アーキテクチャの構成を図 1 に示す.

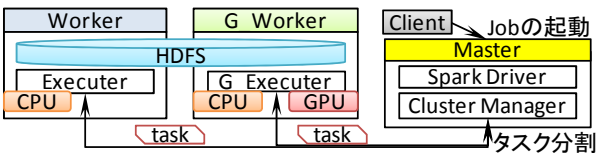

図 1 GPGPU を用いた並列高速化 Spark の構成

G_Worker 内で GPU を用いて高速処理を行う. そのためには Host Memory(以下 H_Memory)上の RDD オブジェクトを D_Memory にコピーする必要がある. しかし, RDD は GPU の D_Memory 上でデータ変換を行うことができない. そのため, メモリへのデータ割り当ての拡張を行う.

4.2. メモリへのデータ割り当ての拡張

Spark 実行時に D_Memory 上に配置するデータを不変データと可変データの 2 種類に分類する. これにより, RDD の特性を活かしかつ D_Memory を効率的に利用ができる. 提案の利用方法を述べる.

(1) 不変データ: 入力データとして利用

RDD から作成される. RDD は読み取り専用のオブジェクトである. そのため GPU で処理を行うために、RDD データから D_Memory へ転送するためのコレクションを H_Memory 上に作成し, コレクションの大きさだけ D_Memory の領域を確保する. その後, コレクションを D_Memory へコピーする. このデータは処理を行うための入力データとして用い, アプリケーションが終了するまで変更されない(図 2).

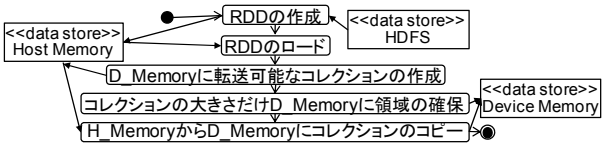

図 2 不変データの生成

(2) 可変データ:出力データ, 入力データとして利用

D_Memory 上に作成される. 事前に記憶領域を D_Memory 上に確保しておき, その領域に GPU を用いた処理結果を格納する. 処理結果は H_Memory にコピーされ, その後 RDD に変換される. 繰り返し処理が行われる場合は同じ記憶領域を用いて処理を行う. 処理を行う際の元データは不変データを用いる. 繰り返し処理を行う場合直近の可変データを入力データとして扱い, 新しいデータが生成された後入力にデータの可変データは削除する(図 3).

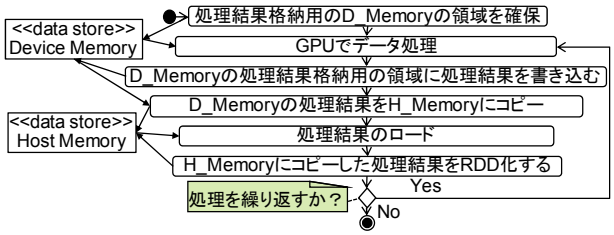

図 3 可変データの生成

4.3. GPU を用いたグラフ並列処理方法

D_Memory 上でのグラフはコンパクトな配列での表現[4]を用いる. これにより容量の少ない D_Memory を効率的に利用する事が可能である. グラフ処理では不変データはグラフに関するコレクション(隣接配

Parallel Processing of Large-Scale Graphs Using Spark on GPGPU

†Yuki Inamoto, Graduate School of Sciences and Engineering, Nanzan University.

‡Mikio Aoyama, Department of Software Engineering, Nanzan University.

列, 各ノードの入出次数)となり, このグラフの情報を用いてグラフ処理を行う. グラフ処理の結果を可変データとして生成し, 処理結果は H_Memory に転送し RDD として格納される. 提案アーキテクチャでグラフ処理を行う場合の処理の振る舞いを示す(図 4).

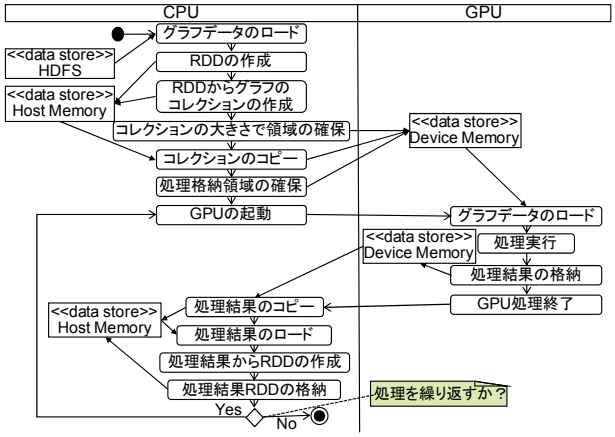

図 4 グラフ処理の振る舞い

5. プロトタイプの構築

提案の評価を行うためにプロトタイプを構築した. 実行マシンの構成を表 1 に示す. 利用マシンは 1 台であるが, 複数スレッドを用いて分散させる. 実装は Python で GPU を用いるために PyCUDA API を用いた. GPGPU を使用した Spark の Python で実装したページランクプログラムの規模は 171(LOC)である

表 1 プロトタイプの構成

OS	Ubuntu 14.04
Spark	1.1.0
CUDA	6.5
CPU	Intel Corei5-4460(4Core,4Thread, Meory:16GB)
Graphic Board	NVIDIA GeForce GTX745 (CUDA Core:384Core, Device Memory:4GB)

6. 例題への適用と評価

6.1. 例題への適用

例題として, 静的ページランクアルゴリズムを, Graph500[3]の Generator で生成した Kronecker Graph に対して適用した. 静的ページランクの繰り返し回数は 50 回行う. 適用した Kronecker Graph のパラメータは SCALE4 から 22, edge factor は 16 とする. これは 2 の SCALE 乗のノード数, (総エッジ数 / 総ノード数)が 16 の有向グラフを作成することである.

6.2. 評価

各 SCALE の Kronecker Graph について, Spark 単体, GraphX, 提案実装でのページランクアルゴリズムの実行時間を比較評価した. 図 5, 図 6 共に横軸は SCALE 数を表す. また, 図 6 の横軸の加速倍率は(CPU の Spark の実行時間 / GPU の Spark 実行時間または GraphX の実行時間)である. 提案および GraphX は Spark 単体より高速化した. 提案実装と GraphX は SCALE15 から 20 にかけて処理時間が提案よりも実行時間が短くなったが SCALE21 以上になると提案のほうが高速化した.

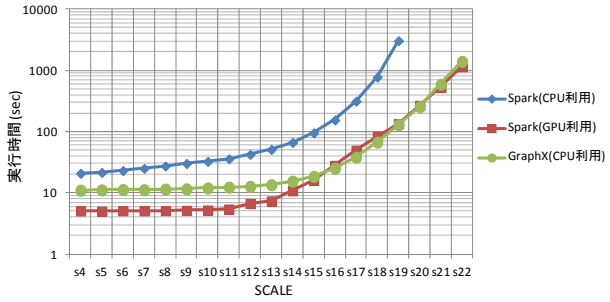

図 5 実行時間

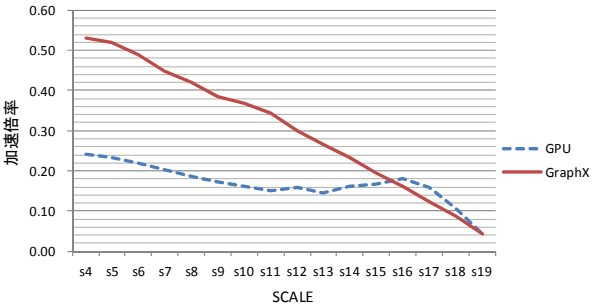

図 6 加速倍率

7. 考察

例題の条件では GPU を用いることで GraphX を利用した場合とほぼ同等の高速化を実現した. 一部 GraphX のほうが高速である箇所がある, これは提案の RDD からグラフのコレクションを作成する時間が原因であると考える. グラフが大規模になると GPU, GraphX 共に高速化の効果が増大する. 今後, D_Memory 上に格納できない大規模グラフに対しては, グラフを分割し複数 GPU を用いて並列処理することが効果的であると考えられる.

8. まとめ

GPGPU を用いて Spark を高速化した. D_Memory 上のデータを分類することで, Spark の RDD の特性を活かしながら GPU での処理の高速化した. ページランクアルゴリズムで提案の評価を行い小規模でも高速化を実現したが, 大規模なグラフであるほど高速化の効果が増大することを確認した.

参考文献

[1] Apache Software Foundation, Apache Spark, http://spark.apache.org/.
[2] H. Bingsheng, et al., Mars: a MapReduce Framework on Graphics Processors, PACT'08, ACM, pp. 260-269.
[3] Graph500, http://www.graph500.org/.
[4] P. Harish, et al., Accelerating Large Graph Algorithms on the GPU Using CUDA, HiPC'07, pp. 197-208.
[5] R. S. Xin, et al., GraphX: A Resilient Distributed Graph System on Spark, GRADES'13, ACM.
[6] K. Shirahata, et al., Hybrid Map Task Scheduling for GPU-based Heterogeneous Clusters, IEEE CloudCom'10, pp. 733-740.