データストリーム処理を用いた変化点検知の実装と GPU による性能最適化

森田 康介† 高橋 俊博‡ 鈴村 豊太郎†‡
東京工業大学† IBM 東京基礎研究所‡

1. はじめに

近年、工場の生産ラインにおける機械の故障やエラーの監視の為に熱や速度を常に計測するセンサーデータの普及や、ボットネットなどサイバーテロの監視に使われるサーバのオンラインデータの観測など、リアルタイムに複数のデジタルデータからの情報を抽出し、変化点・異常検知を実現する需要が高まっている。本稿ではデータストリーム処理システム System S を用いたリアルタイムの異常検知アルゴリズムの実装と GPU による高速化について述べる。

2. 変化点検知のデータストリーム処理と GPU を用いた高速化

本稿では変化点検知のアルゴリズムとして特異スペクトル変換 SST(Singular Spectrum Transformation)[1]を対象にする。

2.1 変化点検知アルゴリズム SST

SST は古典的な変化点検知の一つである。観測される時系列データの背後にあるデータ生成機構を想定した際にその生成機構の構造的な変化の検出を扱うので、特にヘテロな動的システムに非常に適した特徴を持つ。また、他の手法とは異なり特定の確率モデルを仮定しない為、入力時系列の多様性に比較的頑強であり局所解の心配がない。実用例として自動車のセンサーデータの監視やサーバ群のエラー検知などがある。この SST においては特異値分解 SVD(singular value decomposition)が実行時間全体の多くを占める為、ウィンドウ幅を大きく取ると従来の計算環境では実時間内にスコアを得る事が出来ない。

2.2 データストリーム処理系 System S

データストリーム処理とは、多様なセンサーから時々刻々到着する大量データをオンメモリ上で処理することにより、従来のバッチ処理と比較してリアルタイム性及び処理の効率化を実現する計算パラダイムである。昨今、このようなデータストリーム処理を汎用化した並列分散ミドルウェア及びプログラミングモデルの研究開発が活発に行われており、我々は IBM Research によって開発された処理系 System S [4] を用いる。

System S は SPADE [4] と呼ばれるストリーム処理に特化したデータフロー型の言語とそれを支える分散処理系から構成される。SPADE ではデータの流れをストリーム、処理ロジックをオペレータと呼び、これらをデータフローグラフとして表現することで記述する。オペレータは射影処理を行う Functor, ウィンドウ処理を行う Aggregate, センサーデータの受信を行う Source, データの送信を担当する Sink など 10 数種類の組込みオペレータと、ユーザが独自に C++ や Java によって定義したユーザ定義のオペレータ UDOP (User-Defined Operator) から構成される。また、マルチコア、メニーコアで構成されるクラスタ上で稼動し、ユーザが SPADE プログラムのオペレータに属性を付加することによって、オペレータのノード割り当てや複数オペレータの融合などを自動的に行う。

2.3 System S/SPADE による SST の実装

System S/ SPADE による SST 実装のコード断片を図 1 に示す。

```
#define WINDOWSIZE 100

stream InputData(data : Float)
  := Source{file:///input.csv,nodelay,csvformat}

stream SubsequenceData(datalist : FloatList)
  := Aggregate(InputData){count(WINDOWSIZE
),count(1)}[Col(data)]

stream SST(score : Float)
  := UDOP(SubsequenseData)[SSTComputation] {GPU
Enabled=false}
-> node(ComputeNodePool,0)

Nil := Sink{file:///score.out,nodelay,csvformat}
```

[図 1] SPADE プログラム（一部）

Source：センサーからの時系列データを読み込みデータストリームとして排出,

Aggregate：決められたウィンドウサイズ分の部分データをバッファリングしてリスト型のデータストリームに加工,パラメータとしてはウィンドウサイズとスライドさせるサイズを指定(図 1 ではそれぞれ、100 と 1 が指定されている)

UDOP（ユーザ定義オペレータ）：Aggregate に

Implementation of Change-Point Detection by Data Stream Processing and Optimization via GPU.
Kosuke Morita †, Toshihiro Takahashi ‡ and Toyotaro Suzumura † ‡.
† Tokyo Institute of Technology
‡ IBM Tokyo Research

よってバッファリングされたある範囲のデータに対して SST を実行し、変化度合いのスコアを計算。SST における SVD 演算は C++の線形代数ライブラリ newmat10[6] を使用。

Sink：UDOP から受け取ったスコアを外部に出力

2.4 GPU による高速化

前章では C++ の行列演算ライブラリを CPU 上で実行することによって System S の SST を実装したが、SVD は計算量 が $O(n^3)$のため重く、ウィンドウサイズの増加に比例して実行時間も指数関数的に増大する。そのため、応答時間が重要視される変化点・異常検知のアプリケーションにおいては、比較的長期間の異常パターンを検出することが困難になる。我々はこの問題に対して、SVD の計算部分を、近年科学技術計算において様々な高速化が検証されている GPU 上で計算を行うことによって高速化を実現する。

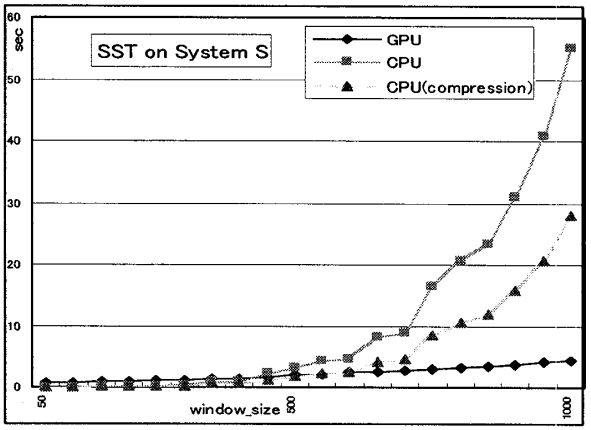
SVD 演算の GPU による高速化にあたっては深谷らなどの研究[3]もあるが、今回は線形代数ライブラリ LAPACK を CUDA (Version 2.3) 上に移植した CULA Basic (Version 1.0) [5]の SVD 関数を用いた。System S 上では、ユーザ定義のオペレータ UDOP において、CULA ライブラリの SVD 演算 を呼び出すように改変することで実装した。但し、SPADE プログラムのレベルでは、図1の SPADE プログラムを再利用することができ、UDOP オペレータに渡すパラメータ GPUEnabled を true にすることで、SVD 計算が GPU 上で行われる。

3. 性能評価

評価環境としては 単一ノード上で計測し、CPU は Athlon X2 (2 コア, 2.6 Ghz), メモリ 4GB, OS は CentOS 5.2(kernel 2.6.18)を用い, GPU は NVIDIA 社の Tesla C1060(240 コア, 1.296 Ghz, 4 GB)を用いた。

SVD の対象となる行列のサイズを 50 から 1,000 まで 50 刻みで SST のスコア算出にかかる時間を計測したところウィンドウサイズ w が 400 の時には CPU : GPU が 0.92 秒 : 1.49 秒であるが、w が 450 の時には 2.21 秒 : 1.68 秒となり GPU の方が高速にスコアを算出している結果が得られた。さらに w が 1,000 の場合では 55.12 秒 : 4.44 秒 と 約 12.44 倍の高速化が実現できた。

GPGPU においてはメモリ転送などのオーバーヘッドが生じるため小さなウィンドウサイズでは CPU に劣るが、 450 以上の大きなウィンドウサイズで SST スコア を算出する場合に GPU は有効と言える。また、グラフより、ウィンドウサイズの増加に伴って CPU 版では計測時間が



450 付近以降は指数関数的に増加しているのに対して、GPU を用いた際には実行時間の増加を抑えることができ、ウィンドウサイズ 1000 においても 4 秒台とアプリケーションによってはリアルタイムな異常検知、変化点検知を実現することが可能になることが示された。

尚、グラフの▲で表されるデータは SVD の行列圧縮[2]を用いて高速化を行った際のデータであるが、この結果に対しても GPU を用いることによる優位性が見て取れる。

4. まとめと今後の課題

本稿では 変化点検知アルゴリズム SST のデータストリーム処理を System S 上で実装し、また更に GPU を用いて高速化を実現した。その結果、ウィンドウサイズ 1,000 の SST の計算において、従来の計算機環境であれば 55.12 秒 を要していた所を GPU で高速化する事で 4.44 秒 に抑え、大きなウィンドウサイズにおいてもリアルタイム性を実現できることを示した。

今後は、現在の CUDA の仕様により、同時に 1 カーネルのみしか実行できない制約の為、センサーデータの 1 次元のみの SST の計測を扱ったが、マルチ GPU の利用や、NVIDIA が開発する複数カーネルを同時実行可能な次世代 GPU (Fermi) によって多次元の同時計算と性能向上を図る。

参考文献

[1] T. Ide and K. Inoue. Knowledge Discovery from Timeseries Data using Nonlinear Trans-formations, The 4th Data Mining Workshop of JSSST, 2004.
[2] T. Ide. Speeding up Change-Point Detection using Matrix Compression. IBIS, 2006.
[3] 深谷 猛, et. al. 正方行列向け特異値分解の CUDA による高速化. HPCS, 2009.
[4] Bugra Gedik, et. al. SPADE: The System S Declarative Stream Processing Engine, SIGMOD 2008.
[5] CULAtools. http://www.culatools.com/.
[6] newmat: Matrix Library in C++.