# ストリーム解析処理再訪　～HPC の観点から～

秋岡明香*

## 発表概要

ビッグデータ解析、特に時系列に沿って高速で到着する大量データの解析処理への需要が高まっている。現状では、解析処理能力や計算規模の問題から、統計的な解析や限られたパラメータセットについてのデータマイニングが限界である。ビッグデータ解析が目指す、世の中の全てのデータに対するリアルタイムな網羅的かつ横断的解析は、現状では非現実的である。

数値計算など、並列分散処理分野で飛躍的な速度向上を継続しているアプリケーションは、LINPACK などの世界標準ベンチマークがその特徴を的確にとらえている。したがって、これらのベンチマークを高速実行する計算環境は、数値計算などのアプリケーションを高速実行可能であると見なすことができ、計算環境の研究開発の指標が具体的かつ明確である。結果として、コミュニティ全体でこれらベンチマークの高速化に向けた研究を進めることができた。一方で、ビッグデータ解析をはじめとする、数値計算以外のアプリケーション利用者には、並列分散処理分野で研究されてきた高速化手法や計算機アーキテクチャは有用でないとの意見も少なくない。

本発表では、ビッグデータ解析で用いられる手法のうち、ストリーム解析処理に対象をしぼり、ストリーム解析処理の挙動解析とモデル化を行なう研究について紹介する。こうした挙動解析とモデル化を行なうことで、既存の計算環境や高速化手法がストリーム解析処理に対して有効かどうかを客観的に判断することが可能になる。さらに、既存の計算環境や高速化手法がストリーム解析処理に適していない場合には、この研究をさらに進め、挙動解析結果とモデルに基づいた、ストリーム解析処理に特化したベンチマークを構築し、計算環境への要求要件や高速化手法研究の明確な指針を示す必要がある。

* 明治大学 総合数理学部 ネットワークデザイン学科

## 質疑応答

Q. データ依存というのはコンパイラが生み出す依存関係のことか？
A. そうではなく、タスクからくるもの。スケッチのところで依存関係が起きる。

Q. ハードウェアの図では特定のアーキテクチャいんついては何も言っていないような感じがするが。
A. いまの一般的なアーキテクチャでいいのではないかということ。

Q. データを切る粒度などはストリーム記述言語で静的コンパイルで決めてもいいのでは。
A. 入力データに計算コストが依存するタスクだと動的解析も必要である。

Q. タスクグラフを使って表現して、グラフ分割は環境に応じて自動的にやってくれる処理系がある気がするが、そういう処理系なら、お金を出せば性能が稼いだりできて嬉しいか？
（StreamIt のことを指していた模様。http://groups.csail.mit.edu/cag/streamit/）
A. そう思う。

コメント：専用ハードウエアに特化していても良ければ、パイプライニングを自動で最適化するためのツールもある。
T. Tanimoto, S. Yamaguchi, A. Nakata, and T. Higashino,
“A Real Time Budgeting Method for Module-Level-Pipelined Bus Based System using Bus Scenarios,” Proc. DAC2006, pp. 37-42, 2006.

## 発表内容

発表内容は次ページ以降のスライド資料に示す。

# ストリーム解析処理再訪
# ～HPCの観点から～

秋岡明香（明治大学）
akioka@meiji.ac.jp

## 動機

- ビッグデータ時代の到来
- 「HPCの人達、なんにもわかってないよね」という意見を聞くこと多数
  - 「HPCで培って来た高速化技術は我々のアプリケーションには全く役に立たぬ！」
- 本当にわかってないのか…？
  - 何か認識がずれている？
  - 要求事項が理解できていない？

## 研究の目的

- ストリーム解析処理の挙動解析とモデル化
  - データインテンシブとは根本的に何が違うか
- ストリーム解析処理のベンチマーク化
  - 将来課題
  - 本当に挙動が異なるならばLINPACKのようにベンチマーク化するべき
  - ベンチマークで注力点が明確になれば、アプリケーション最適化もハードウエア開発も格段に進みやすくなるはず

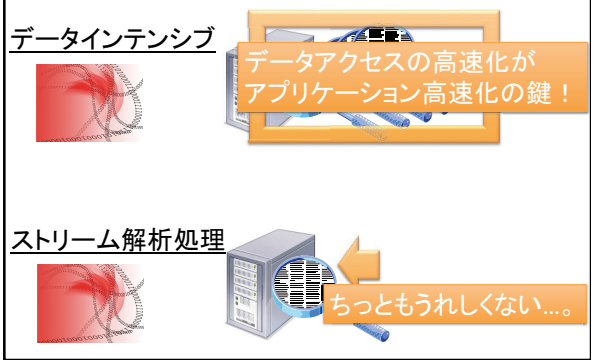
## ストリーム解析処理 (1)

- 大規模データの収集、蓄積
  - 数千台の分散HWに対応した、スケーラビリティのあるデータ収集蓄積技術
  - ソーシャルグラフを最小カット分割し、複数サーバに分散配置することで、サーバ間アクセスを削減する技術
- 大規模データの加工
  - データを蓄積すること無く情報処理を行うストリームプロセッシング、オンライン機械学習技術
  - データ分析用のクエリ履歴から、中間データを自動に事前生成し処理時間を削減する技術

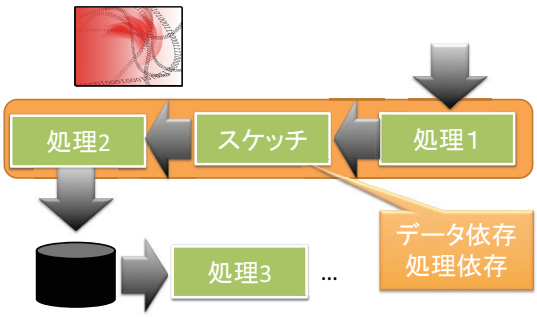
## ストリーム解析処理 (2)

- frequent countのようなアルゴリズムを想定
- アプリケーションの挙動が異なることはRaicuらが指摘済み
  - Raicu, I., et al.: The Quest for Scalable Support of Data Intensive Workloads in Distributed Systems. Proc. the 18th ACM International Symposium on High Performance Distributed Computing (HPDC2009), pp.207-216 (2009).
  - データインテンシブは「write-once-read-many」対してストリーム解析は「write-once-read-once」

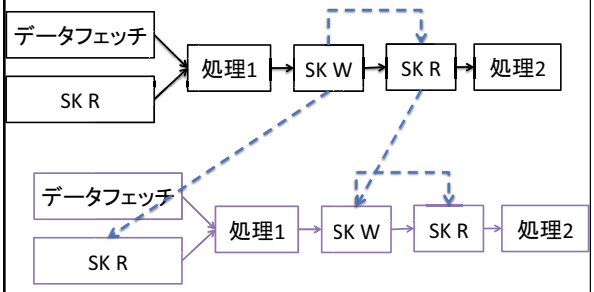
## データアクセスパターンの違い



## ストリーム解析処理の一般モデル (1)



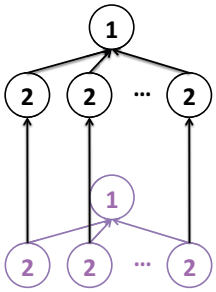

## ストリーム解析処理の一般モデル (2)




## タスクグラフ

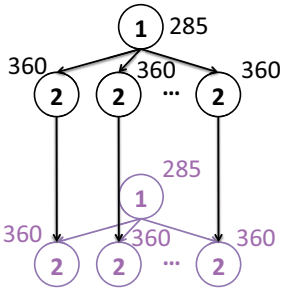
- アプリケーションのデータ依存や制御依存、計算コストなどを模式化した有効グラフ
- スケジューリングアルゴリズムの評価時など対象アプリケーションとして擬似的に使用
- 基本的には既存のタスクグラフと同様の定義
  - ストリーム解析独特の拡張が少々

## データ依存グラフ



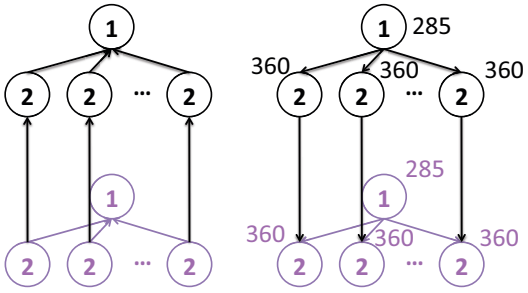

- プロセス間の依存関係が生じるので最小セットを記述
  - 黒:先行プロセス
  - 紫:後続プロセス
- 並列度は静的に決まらないことが多い
- 並列度は実行前に決まらないことが多い

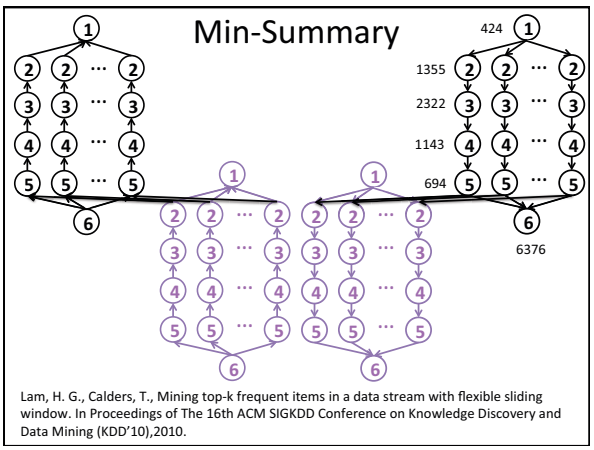
## 制御依存グラフ



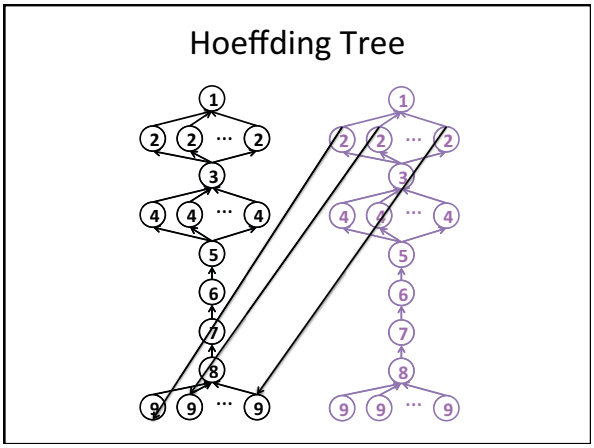

- プロセス間の依存関係が生じるので最小セットを記述
  - 黒:先行プロセス
  - 紫:後続プロセス
- 並列度は静的/実行前に決まらないことが多い
- 通信コストは未対応
- 実行コストについても不確定要素が多い
  →後述

## Naïve Bayesの学習過程





Lam, H. G., Calders, T., Mining top-k frequent items in a data stream with flexible sliding window. In Proceedings of The 16th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'10),2010.




## 実行時間見積りに関して

- 計算対象のデータ量等で実行時間の大きな変動が起こりうる
  - Min-Summaryの閾値再計算など
- 平均値による実行コストの表現は意味を持たなくなる場合がある
  - 実行コスト見積りについて別軸の表現が必要

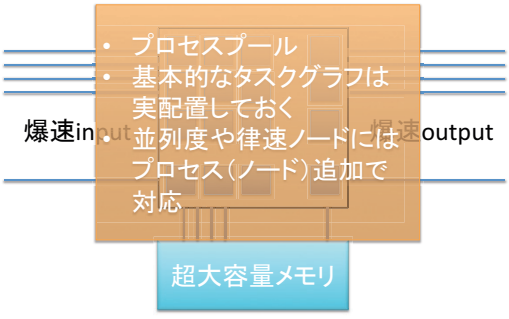
## とうとうtoo many coresの時代到来？

- 以上のタスクグラフは個別の手法の模式化
- 実際の運用では、個別手法の結果に応じて次に適用すべき個別手法が変わるはず
- 並列度問題、実行コスト見積り問題もある
- 各ステージの実行コストも非常に小さい
- too many coresの中で完結してしまえば楽！

## 研究の動機に戻って…

- 大規模ストリーム解析処理は既存のHPCな技術で解決ができるか？
  - 「No」という材料は見当たらない
- 「ビッグデータ」の目的ってそんなもの？
  - 決まりきった処理をするのではなく、総当たりに調べるとかの方がロマンが…。
  - そういう力技に任せた単純作業こそ、コンピュータにさせるべき。
  - 「計算が追いつくならば、もっと複雑な手法も使いたい」

## こんな感じ？



## それでもやるべきことはある。（想像が及ぶ範囲編）

- 各プロセスが状態を保持するので、同じ手法でも複数セット持つ必要がある可能性も
  - ひとつのダイでは当然足りなくなる
- 結合が密な計算ノードを同じダイに配置する必要性
  - 計算ノードの粒度自体の検討も必要
  - 小データの送受信が頻出する可能性
  - 動的再配置は現実的なのか？
- もちろん、個別の手法の実装上の工夫も不可欠

## それでもやるべきことはある。（想像が及ばない範囲編）

- データの送受信や依存関係を軸にしたコーディングや手法検討はフレンドリーではない（と思う人は世の中に多いらしい）
  - Stream programmingはアプリ開発者を楽にしているか
- 目視による手法検討を捨ててタスクグラフ生成をコンパイラ等に任せると、高速化にも限界が？
  - 「職人技」に落ちては広い普及は難しい

## ちゃぶ台をひっくり返す疑問

- ストリーミング処理とバッチ処理の厳密な区別は本当に必要？
  - 個別の手法が充分に速く、バッチ間隔が充分に短ければ？

## 関連研究 (1)

- 疑似ランダムタスクグラフの生成や生成手法に関する研究は多数
  - Task Graphs for Free (TGFF)
  - GGen
  - Task Graph Generator
    - FFT、ガウス消去法、LU分解は実装からのタスクグラフを公開
  - Standard Task Graph Set
    - 疎行列ソルバ、ロボット制御プログラム、SPEC fppppは実装からのタスクグラフを公開

## 関連研究 (2)

- Cordeiroらはタスクグラフの出来の重要性を指摘
  - 生成手法が異なる2つのランダムタスクグラフを用いて特定のスケジューリング手法を評価したところ、3.5倍の速度向上を実現
  - Corderiro, D., Mounie, G., Perarnau S., Trystram, D., Vincent, J. M., and Wagner, F.: Random Graph Generation for Scheduling Simulations, Proc. the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools'10) (2010).

## まとめ

- ストリーム解析処理に特化したベンチマーク生成に向けて、ストリーム解析処理の分析、モデル化を開始
  - 一般モデルとの大きな齟齬は今のところはない
  - さらに使用頻度が高い手法について解析とタスクグラフ化を進めていく
    - コンパイラの助けを得ることで作業は圧倒的に進みが速くなる、が。
- 実行時間見積りや並列度の扱いに課題