【VLDB2010勉強会】

Session 20: Databases on Modern Hardware

担当：山室健（NTT）

紹介する2本の論文

- 1. Louis W. (Eidgenössische Technische Hochschule Zürich, Switzerland) et al.: Complex Event Detection at Wire Speed with FPGAs
- 2. Wenbin F. (The Hong Kong University of Science and Technology, People's Republic of China) et al.: Database Compression on Graphics Processors
- 3. LoggingRyan J. (Carnegie Mellon University, United States of America) et al.: Aether: A Scalable Approach to Logging

Database Compression on Graphics Processors

- 論文の概要
  - 近年DBのクエリ処理最適化に適用されることが多くなったGPUだが、**メインメモリ-GPUメモリ間のデータ転送がボトルネック**になる問題[18][19]
  - 本論文では上記の問題に対して、9つのデータ圧縮手法を実装し、これらを任意に組み合わせること(cascade-compression[15])で、より効率的な圧縮を行い、データ転送ボトルネックを緩和
  - さらに、**最適な圧縮手法の組み合わせを発見**するプランナも合わせて設計
  - GDB[18]とMonetDB[6]に実装し、人工データとTPC-Hを用いて評価
- 現状の課題
  - GPUを使用したクエリ処理の**15-90%がメインメモリ-GPUメモリ間のデータ転送**処理であることが指摘[15][16]

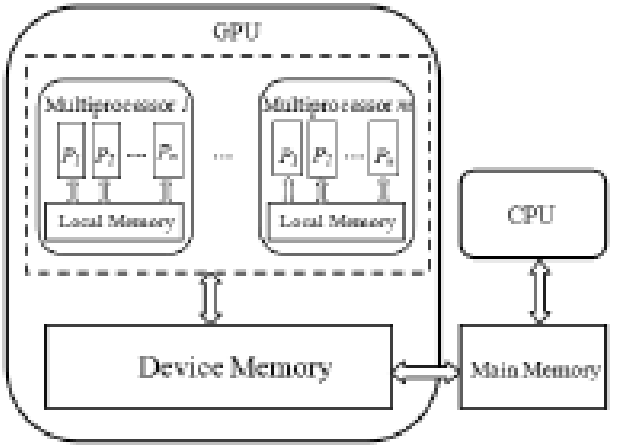


Figure 1: The GPU Many-core Architecture.

Database Compression on Graphics Processors

- 論文で利用している9個の圧縮手法
  - Main Schemes – データサイズを小さくする手法
    - 固定長/可変長のNull Suppression、辞書圧縮、ビットマップ圧縮、ランレングス圧縮
  - Auxiliary Schemes – 圧縮のためのデータ変換に関する手法
    - Frame-Of-Reference、Delta、Separate、Scale
- 最適な組み合わせ＆順序の決定を行う圧縮プランナの設計
  - 探索空間が大きくなるため、圧縮対象の特性を考慮に入れて**探索空間の縮退**
    - ソートされていれば、圧縮しやすいなどの枝狩り
  - 圧縮率、解凍速度などを考慮に入れた評価関数の定義
    - 解凍速度に関しては、GPU内の処理量と稼働時間を考慮にいれて実行時間を推定するモデルを採用

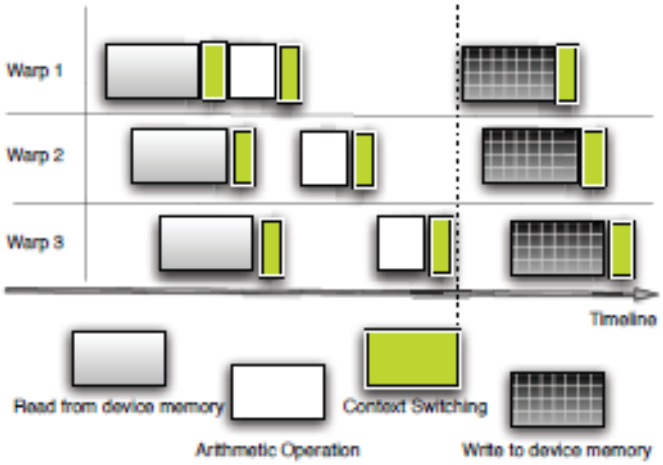
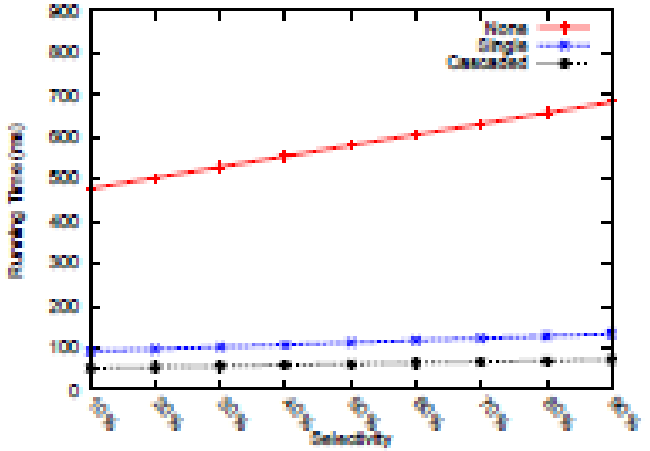
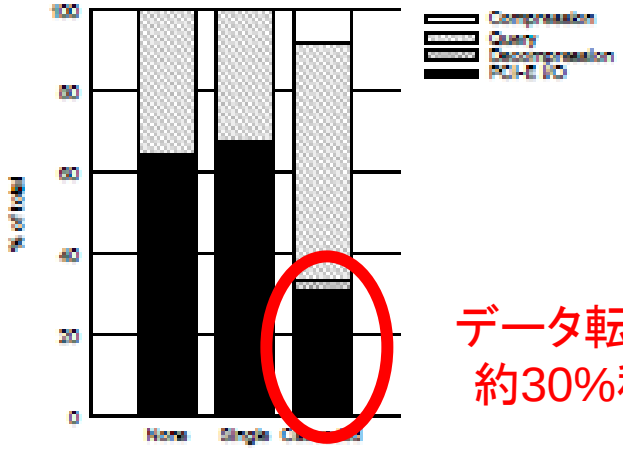


Figure 2: Warp scheduling on a multiprocessor.

Database Compression on Graphics Processors

- 人工データを利用したGDB上での評価
  - 非圧縮（下図(a)の赤）、単一圧縮手法（下図(a)の青）、圧縮プランナが生成したcascade-compression（下図(a)の黒線）の3つで応答速度を評価
- 他にもMonetDB上での評価が実施
  - データ全体を解凍する必要があるTable-scanでは性能悪化
  - 部分的な解凍が必要な処理の場合には性能向上→ただしCPU上で同等処理をさせた場合と大差なし





データ転送の割合が
約30%程度に減少

(a) Overall table-scan performance. (b) Time breakdown of GDB table-scan, with selectivity 40%.

Aether: A Scalable Aproach to Logging

- 論文の概要
  - 大量の小さな更新Tr.が入力されるワークロード（TPC-B相当）が単体のマルチコアマシン上で動作するDBを想定
  - 上記の環境で発生する4つのボトルネックを明らかにし、これらを解決する手法を提案
  - 従来手法と比較し、20-69%の性能改善を達成
- DBのログマネージャにおける4つのボトルネック
  - 1. 大量の小さな更新I/O
  - 2. Log Flushの際のロック期間
  - 3. ブロックI/Oによるスレッドのリスケジュール
  - 4. メモリ内ログデータ構造のアクセス直列化

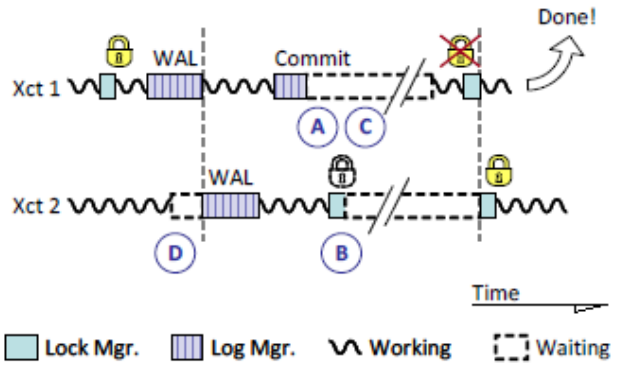


Figure 1. A timeline of two transactions illustrating four kinds of log-imposed delay: (A) I/O-related delays, (B) increased lock contention, (C) scheduler overload, and (D) log buffer contention.

Aether: A Scalable Aproach to Logging

- “4. メモリ内ログデータアクセスの直列化”に対する3つの提案手法
  - (C) Consolidating Buffer Allocation
    - Consolidation Arrayという手法を利用して[9][15][19]、連続するログレコードの挿入を1つにグループ化
  - (D) Decoupling Buffer Fill
    - ログレコードのCOPY処理は本質的には直列化が必要な処理ではないため、Tr. のCritical Pathから外すことで、レコードサイズの影響を除外
  - (C) + (D)
    - 互いの手法は排他的ではないため
      2つを合わせたハイブリッドな手法

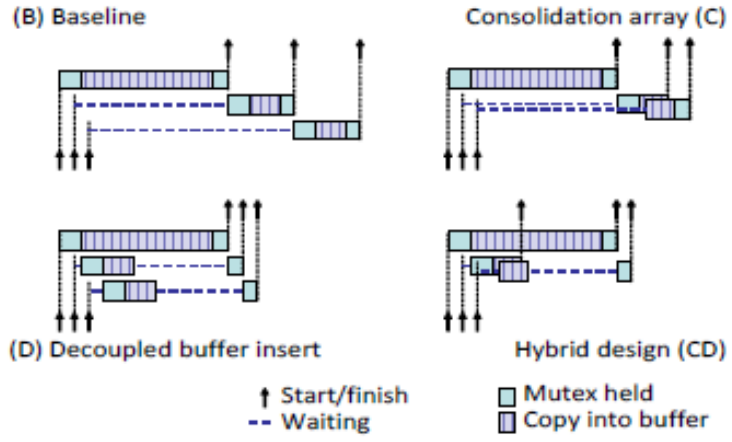


Figure 6. Illustrations of several log buffer designs. The baseline system can be optimized for shorter critical path (D), fewer threads attempting log inserts (C), or both (CD)

Aether: A Scalable Aproach to Logging

- 同時スレッド数とログサイズを変化させた場合のスループットを評価（左が同時スレッド数、右がログサイズ）
- 各手法の動機が反映された結果に
  - Cの手法は、スレッド間のロックContentionに対する手法
  - Dの手法は、ログサイズの影響を小さくするための手法
- ログサイズを大きくすると、提案手法は全てメモリハンド幅上限に到達

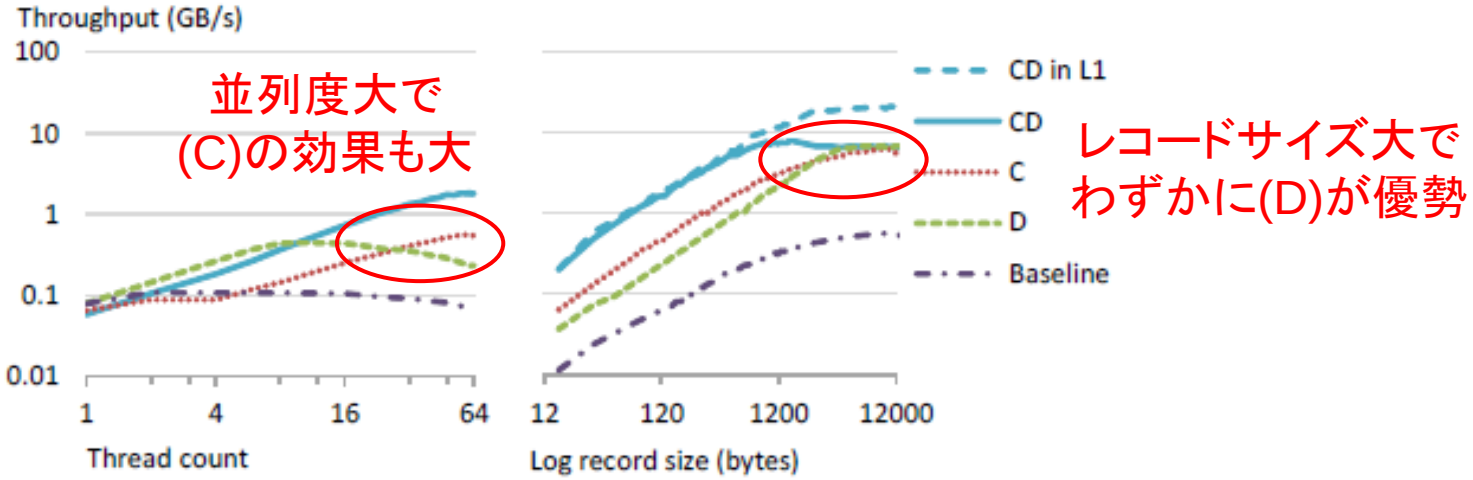


Figure 8. Log buffer scalability with respect to thread counts (left, 120B log records) and log record size (right, 64 threads)