マルチGPU環境におけるCRS形式疎行列・ベクトル積の入力行列の最適化による高速化

岡田 和人(電気通信大学)，岡本 吉央(電気通信大学)，今村 俊幸(理化学研究所 計算科学研究機構)

研究目的

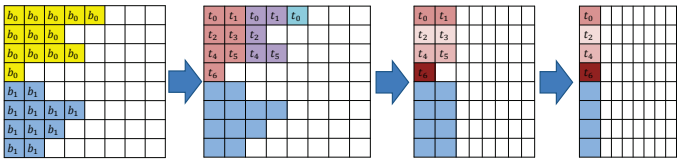
- nVidia社製GPUのGPGPU開発環境CUDAにおけるより高速な疎行列・ベクトル積の実装
- より大規模な疎行列への対応
- 複数のGPUを搭載した計算機への対応

CUDA向けCRS形式疎行列・ベクトル積実装

- CRS形式疎行列格納方式

CRS形式は疎行列を行優先で一次元配列に格納
同時にその要素の列の場所を同様に格納
その一次元配列内で行の一番最初の要素にあたる要素の添字を一次元配列で格納

- CUDA向けCRS形式疎行列・ベクトル積実装概要（既存研究）



1ブロックが複数行を計算
各行の部分積をスレッドごとに計算
求めた部分積の総和を行ごとにリダクション計算
各行の部分積の総和をベクトル積として格納

CUDAについて

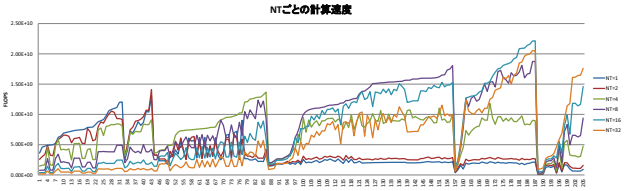
- カーネル関数：GPUで動作する関数
- スレッド：カーネル関数を実行する最小単位
- ブロック：スレッドをまとめたもの
  - 同一ブロック内のスレッドならばデータを共有することができる
  - 一般にブロック間でデータの共有は不可
- グローバルメモリ：GPUのメインメモリ
- シェアードメモリ：同一ブロック内のスレッドでデータを共有できるメモリ
  - 小容量だがグローバルメモリの100倍高速

実験環境

- OS：Fedora18
- CPU：Intel Core i7
- メモリ：16GB
- GPU1：GeForce GTX 590
- GPU2：GeForce GTX 590
- コンパイラ：Nvidia CUDA Compiler Ver.5.5

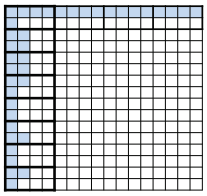
既存研究の性能と課題

既存研究の実装について調べるため，約200の疎行列に関して一行当たりの計算スレッド数（NT）を変えながら計算速度を測定した．
実験結果から，一番高速に計算できたNTごとにまとめて結果を示す．
なお，1ブロック当たりのスレッド数は256スレッドとする

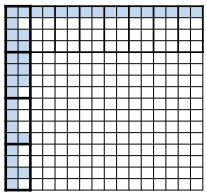


上図においてどのNTでも計算速度が落ちている行列について調べた．その結果，以下の2つの傾向が見られた．

一行ごとの比例要素数の最大値と平均値に差がある場合



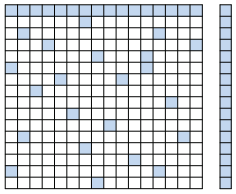
NT=4のとき，一行目の計算は4回のループで終了するが，同時に計算できるのは8行しかない．



NT=2のとき，一度に16行計算できるが，一行目の計算に8回のループが必要となる．

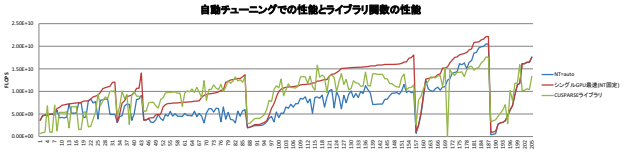
GPU上で4ブロック，32スレッド同時に計算できるとする．
左図は疎行列の非零要素を左詰めにして示したものである．
色のついたセルが非零要素で，太枠は1ブロックで同時に計算する要素を示し，中枠はブロック内でループ処理を行う要素である．
NTが固定である場合，これ以上適切な割り当てができない．上図のどのNTでも計算速度が上がらない部分の行列がこれにあたる．

列インデックスが分散している場合



左図は色付きのセルが非零要素である疎行列とベクトルである．
例えば左図の疎行列・ベクトル積の場合，メモリ上で左の疎行列と右のベクトルにアクセスする．CRS形式の疎行列・ベクトル積では，行ごとに積和演算をするので，このように疎行列の列インデックスにばらつきがあると計算する行が変わるたびにベクトルの別の要素にアクセスすることになる．したがって，キャッシュの利点が生かせなくなるため，計算速度が上がらない．

また，参考文献1に従ってNTを $\max(1,\min(32,2^{\lceil \log_2(\frac{nonzero}{row}) \rceil}))$ とした場合の計算速度と，上記の単一GPUにおける最速の値，CUSPARSEライブラリ関数の速度との比較を示す．



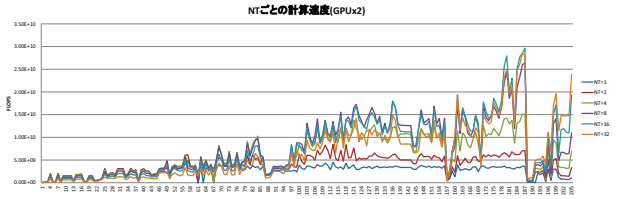
単一GPUでの最速の値と，自動でNTを定めた場合を比較すると，自動で定めた場合は最適なNTを選択できていないことがわかる．また，最速となるNTを選択したとしてもCUSPARSEライブラリの関数より速度が出ない場合がある．

考察とこれからの課題

- 半分以上の行列で単一GPUにおける計算よりも性能が落ちた
  - 複数のGPUを用いることによる同期やメモリ管理の問題
- 一行の零要素数順にソートして計算しても逆に遅くなった
  - 今回用いたチューニングの方法に課題が残る
- Keplerアーキテクチャ上における検証

既存研究の複数GPUにおける性能

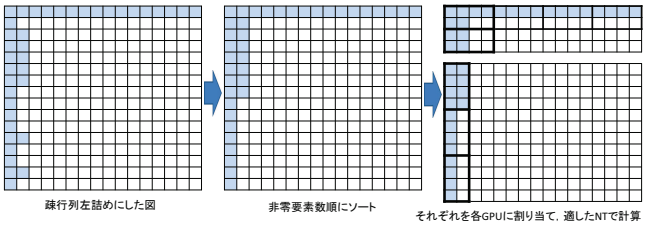
複数のGPUで実装した同様の計測結果を示す．GPUが複数になると，タスクを分割しなければならない．ここでは単純に，行列の上から非零要素が半々になるように上下に分割した．



GPUが増えると，GPU間の同期やデータの転送などに時間がかかり，計算量が小さくなるほど計算速度が落ちている．逆に計算量が増えるほど，計算速度が上昇する傾向がわかる．
単一GPUで計算した時とNTの計算速度の傾向と異なるのは，もともと行ごとの非零要素数に偏りがあり，計算する疎行列を分割した際に非零要素数に差が生じてしまったものと考えられる．

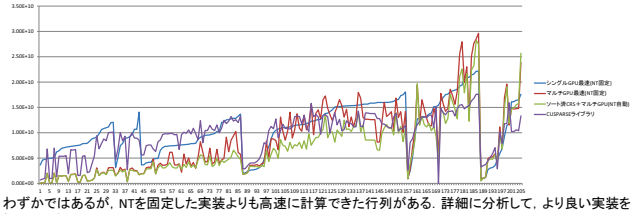
解決策

一行の非零要素数の最大値と平均値に大きな差のある疎行列に対して，以下の方法で入力行列を整理することにより，複数のGPUを用いた環境での高速化，性能の安定化を試みた．



疎行列左詰めにした図
非零要素数順にソート
それぞれを各GPUに割り当て，適したNTで計算

上記の手順により，疎行列データを計算しやすい形にしてから計算を行う．
計算部分は既存研究のものをもとに実装して用いることとする．
なお，適したNTの定め方も参考文献1より，$\max(1,\min(32,2^{\lceil \log_2(nonzero/row) \rceil}))$とした．
以下に各疎行列のシングルGPUにおける最速の結果とマルチGPUにおける最速の結果，CUSPARSEライブラリの結果とあわせて，ソート済CRS形式疎行列・ベクトル席の計算速度を示す．



わずかではあるが，NTを固定した実装よりも高速に計算できた行列がある．詳細に分析して，より良い実装を探っていきたい．

参考文献

1. GPUにおける高速なCRS形式疎行列ベクトル積の実装，椋木 大地，高橋 大介，情報処理学会研究報告 Vol.2013-HPC-138 No.5，2013
2. CUDA BY EXAMPLE　汎用GPUプログラミング入門，Jason Sanders，Edward Kandrot，株式会社インプレスジャパン，2011
3. GPUプログラミング入門 CUDA5による実装，伊藤 智義，講談社，2013