2K-7

LP ライブラリの GPU 高速化におけるデータ依存性の解析

津金 佳祐† 廣津 登志夫†
法政大学情報科学部†

1. はじめに

近年，CPU の 1 つのコアに対する演算性能の向上の限界により様々な処理性能の向上技術が提案，開発されている．その技術として画像処理を専門に行うハードウェアである GPU を汎用演算に利用する GPGPU がある．このような高速化の対象として考えられる計算処理の一例として，線形計画法(LP ： Linear Programming)が上げられる．線形計画法が解く問題である線形計画問題は，条件や制約を線形等式，または線形不等式に定式化を行いモデル化し，目的関数と幾つかの制約条件で構成した問題である．この線形計画法のライブラリの一つである GLPK を解析したところ，制約条件の係数行列中の要素分布によって関数の実行時間の傾向が異なることと，処理中の中間メモリの必要量が変わることがわかった．そこで本論文では，限られたメモリ量で計算を行う GPU での高速化を目指し，様々な係数行列に対する GLPK ライブラリの挙動解析の結果と，適切なメモリ量の推定について述べる．

2. 線形計画問題

企業等でさまざまな制約条件下での利益や効果の最大化や，費用や時間の最小化の問題を，制約条件や目的関数を線形等式，または線形不等式に定式化し，モデル化した問題である．

$$\begin{aligned} &\text{Max} : x_0 = \boldsymbol{cx} \\ &\text{S.t} : \boldsymbol{Ax} \leq \boldsymbol{b} \\ &\boldsymbol{x} \geq 0 \end{aligned} \tag{1}$$

$$\begin{aligned} \boldsymbol{c} &= (c_1\ c_2\ \cdots\ c_n) \\ \boldsymbol{b} &= (b_1\ b_2\ \cdots\ b_n)^T \\ A &= \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \\ \boldsymbol{x} &= (x_1\ x_2\ \cdots\ x_n)^T \end{aligned} \tag{2}$$

x_0の値を最大(または最小)にするx_1からx_nの値を求める問題である．GLPK は，制約条件を等式にするために足すスラック変数を基底変数，それ以外の変数を非基底変数とし，次の 2 つのアルゴリズムを適応した，線形計画問題を解くライブラリである．

2.1. 改訂シンプレックス法

線形計画法の 1 つである改訂シンプレックス法は，シンプレックス法の反復処理に要する演算数を少なく，かつ計算誤差を減らしたアルゴリズムである．係数行列全体を辞書と置くとシンプレックス法では，辞書全体で演算を行うのに対し，改訂シンプレックス法ではその一部のみに演算を行うため計算量に大きな差が生じる．

2.2. 最大傾斜法

改訂シンプレックス法は，多面体の頂点から枝を経由して目的関数が増加する頂点へと移るアルゴリズムである．そこで最大傾斜法を用いて目的関数と経由する枝の角度が最も小さい方向のピボット列を選択することにより，処理中でのピボット選択回数を減らす．式(3)の根号の中の値を最大傾斜係数と呼ぶ．

$$\cos\theta_j := \frac{\bar{c}_j}{\sqrt{1 + \sum_i \bar{a}_{ij}^2}} \tag{3}$$

3. GLPK の解析

まず初めに，プロファイラである google-perftools を用いて GLPK 中の関数毎の実行時間を測定した．これにより，処理時間の大きな関数についてより詳細な解析を行うこととした．このプロファイルの結果，最大傾斜係数の更新を行う update_gamma 関数と，非基底行列を用いた連立一次方程式を解くための LU 分解を行う eliminate 関数の実行時間が大きい事がわかった．

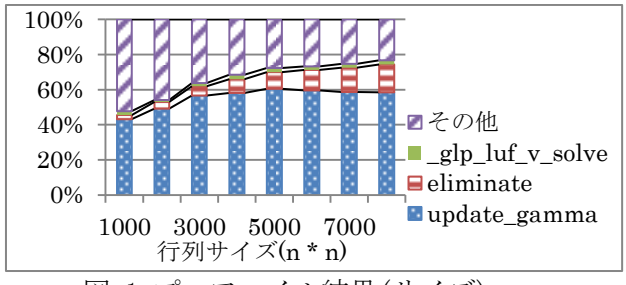

図 1 プロファイル結果(サイズ)

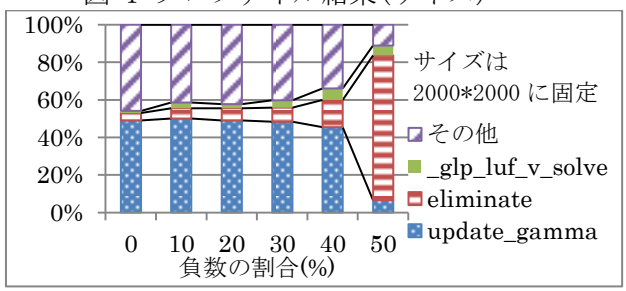

図 2 プロファイル結果(負数の割合)

Analysis of Problem Dependency of LP Library for GPU Acceleration

† Keisuke Tsugane ・Hosei Univ.
Toshio Hirotsu・Hosei Univ.

3.1. eliminate 関数

eliminate 関数で用いられるメモリ量は図 1，図 2 の実験結果を見ると，配列の大きさよりも配列中の数値の分布による依存性が大きそうであるという予想が立つ．そこで，係数行列の要素について，正・負・零のそれぞれの個数を数えメモリ量推定式を立てた．

この関数中で LU 分解の結果を保持する中間メモリを使用している．GLPK の実装では計算の対象となる配列サイズからの概算でこのメモリ量を求めているが，計算結果に対してメモリ量が不足した場合，再度計算し直すというオーバヘッドがかかっていた．この中間メモリの大きさを大きくすれば，無駄な再計算は回避されるがむやみに大きくするとメモリ資源を浪費し，問題規模を大きくすることが難しくなる．そこで，問題となる係数行列の構造による依存性を解析し，問題に適正な大きさを確保する必要が生じる．

3.2. update_gamma 関数

図 3 の最大傾斜係数を保持する配列 gamma は，それぞれの列の係数に対して GLPK 独自の重みを掛けた総和を求め直す．そのため値の更新の競合が起こらず GPU 並列化が可能である．

```
for 1 to non-zero row componets{
  j = get col index
  calculate beg and end;
  for beg to end
    calculate inner product s;
  gamma[j] = gamma_calc(s);
}
```

図 3 update_gamma 擬似コード

3.3. 設計

並列計算を行うワークアイテムの総数を 1 重ループ目のループ数に 32 を掛けた値とし，1 重目のループ内の処理 1 回分を GPU 上の 32 ワークアイテムに行わせる実装を行う．2 重ループ目のベクトル積を求めるために[2]の手法を用いて，図 4 のように 32 ワークアイテムを用い，総和を求める．また，総和の中間結果をローカルメモリに保存を行い，半分ずつのワークアイテムを用いて総和を求める．

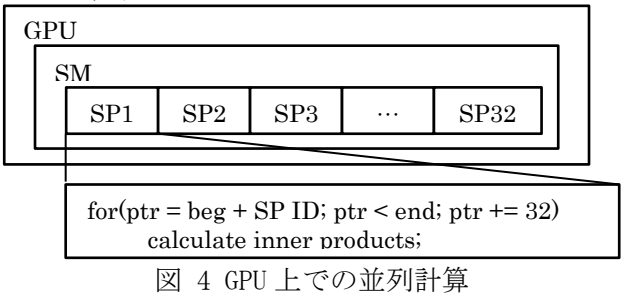

図 4 GPU 上での並列計算

4. 実行結果

GPU 化とメモリ最適化を行った際の元の処理に対する実行時間の短縮をパーセント表示で表した．グラフは上から係数行列サイズの変更，行列サイズを 2000*2000 に固定した場合の負数の割合の増加となっており，結果は GPU 化とメモリ最適化に加え，この 2 つを組み合わせた手法を表す．

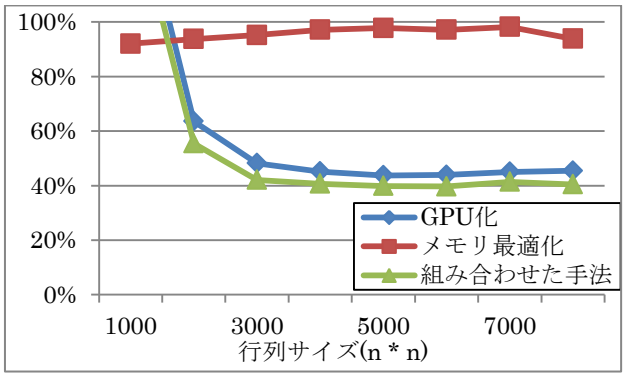

図 5 実行結果(サイズ)

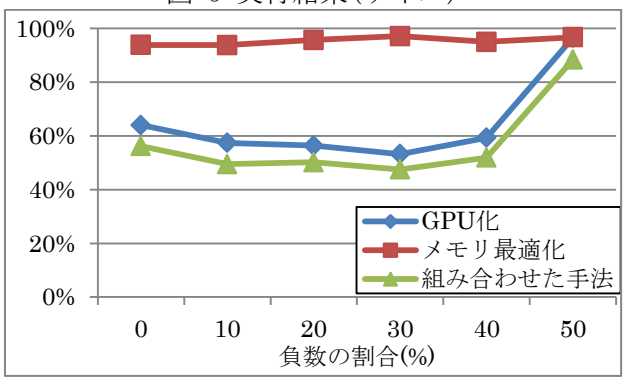

図 6 実行結果(負数の割合)

5. まとめ

結果として，update_gamma 関数の割合が少ない，係数行列中に負数が多い場合では，元の実行時間に比べ約 90%まで，また，係数行列のサイズが小さい 1000*1000 の場合では元の実行時間よりも遅い結果となったが，それ以外の場合では最大で約 40%まで実行時間を少なくすることができた．

参考文献

[1] Mohamed Esseghir Lalami, Vincent Boyer, Didier El-Baz, Efficient Implementation of the Simplex Method on a CPU-GPU System, 2011 IEEE International Parallel & Distributed Processing Symposium, pp.1999-2006, May 16-20, 2011

[2] Nathan Bell, Michael Garland, Efficient Sparse Matrix-Vector Multiplication on CUDA, NVIDIA Technical Reposrt NVR-2008-004, December 2008