Vol.2015-HPC-148 No.15
2015/3/2

有限要素法によるポアソン方程式のGPU向きSpMV計算

三浦 慎一郎[1,a)]　高橋 秀朗[2,b)]　角田 和彦[3,c)]　市川 周一[4,d)]　藤枝 直輝[4,e)]

概要：3 次元ポアソン方程式に対し有限要素法による離散化を行い，ポアソン方程式に対して共役勾配法を適用し，その反復計算の中で最も計算コストの掛かる SpMV 計算において，GPU を用いて高速化を図ることを目的とする．メモリバンド幅の大きい GPU の特性を引き出すため，疎行列の格納に Compressed Diagonal Storage(CDS) を適用した．その結果，GPU による行列ベクトル積の演算性能の最大値は，倍精度浮動小数演算で 56GFLOPS の性能を得た．またメモリバンド幅も 245[GB/s] となり，理論性能値の 85%に達した．これらの性能の要因として，CDS 形式が行列やベクトルでの間接参照のない格納方法であることや，ループアンローリングが可能であること，また GPU の Kepler アーキテクチャでの Read-only データキャッシュの利用などが要因である．さらに有限要素法からなる行列の対称性を利用し，メモリ削減とそれに伴うループ削減が演算性能への影響に最も効果的であった．

1. はじめに

数値流体解析を行う上で，流体の運動方程式および連続の方程式に対して非圧縮性を考慮すると，その微分方程式の解の性質は圧力に関する楕円型，拡散項による放物線型，移流項による双曲型の解の性質を持つ．このうち楕円型の性質を有する圧力項では無限大の解の伝播速度が要求されるため，時間積分を行う上で有限の時間刻み幅を設定して解析することは困難となる．したがって圧力と流速に関して分離解法を行い，圧力項を時間的に陰的に離散化しポアソン方程式として解く方法が一般的な解法となる．また，乱流解析などでは流れの境界層が非常に薄くなる場合や物体回り流れでは流れが急激に変化する場合などに格子幅を十分小さくとる必要がある．この場合にも解の伝播速度の速い放物型性質を有する拡散項も陰的に扱い連立方程式を解く方が，解の安定性や精度，また時間刻み幅を大きくとれるという点で結果的に計算時間の短縮になる場合もある [1].

[1] 東京都立産業技術高等専門学校
Tokyo Metropolitan College of Industrial Technology, Shinagawa, Tokyo 140–0011, Japan
[2] （株）クレオソリューション
CREOSOLUTION CO.,LTD.
[3] 日本大学
Nihon University
[4] 豊橋技術科学大学
Toyohashi University of Technology
[a)] miu@s.metro-cit.ac.jp
[b)] takahah@creo.co.jp
[c)] kakuda.kazuhiko@nihon-u.ac.jp
[d)] ichikawa@tut.jp
[e)] fujieda@ee.tut.ac.jp

このようなことから数値流体解析手法として一般的な差分法や有限要素法などを適用した場合，陰解法を用いる場合が多く，流れ場の状態に応じた計算規模で連立方程式を解く必要がある．

空間の離散化において，ポアソン方程式を差分法や有限要素法などによる離散化を行った場合，その行列はゼロ成分を多く含む疎行列かつ帯行列となる．さらに有限要素法のうち，最も一般的なガラーキン型有限要素法においては係数行列が対称行列となる [2]．この対称性を利用することで，行列記憶の削減を行うことができる．離散化された連立方程式の求解では，共役勾配法（Conjugate gradient method; CG 法）などをはじめとするクリロフ部分空間の反復法によって求められることが多い [3]．このとき行列ベクトル積が計算コストの大半を占める．またこの行列ベクトル積ではメモリバウンドな問題となり，メモリバンド幅の大きな計算機が有効である．

そこで本研究では，汎用な CPU に比べメモリバンド幅の大きい GPU(Graphics Processing Unit) に注目し，疎行列ベクトル積（Sparse Matrix and Vector multiply; SpMV）の高速化を図ることを考える．

これまで GPU を用いた疎行列の格納方法として Compressed Row Storage(CRS)（または Comressed Sparse Row(CSR)）と呼ばれる形式が良く知られている [4]．この方法は疎行列の性質によらず行列成分の格納が可能であるため，汎用性が高く，並列性も良いことから一般的に良く使われる方法である．また完全にゼロ成分を排除して，非ゼロ成分のみを記憶できる点で優位性がある．これは NVIDIA 社が提供する cuSPARSE ライブラリの中でも提

供されている [5]．しかしこの方法は内側ループが小さくベクトル化効率が効かないことや，入力ベクトルにおいてメモリの間接アクセスを必要としており，この影響により計算機アーキテクチャや計算規模によっては性能を引き出すことができない場合がある．このほか CRS 形式の特性について詳細な説明が述べられている文献 [6] がある．

ループのベクトル化を改善する方法として，内側ループで行列の行方向にロードするようにした ELLPACK[7] や Jugged Diagonal Storage(JDS) 形式 [8] が知られている．これは入力ベクトル分の長いベクトルをロードすることができ，ベクトル計算機によるベクトル化効率に有効である．さらに GPU への適用も有効性が示されている [9]．しかし入力ベクトルの間接参照が依然として残る．

これらを解消した方法が CDS 形式である．内側ベクトルの大きさは入力ベクトル分に一致し，入力ベクトルのロードアクセスも連続的なものとなる方法である．したがってこの方法はベクトル計算機に対して有力な方法 [10] である．

GPU を用いた CDS 形式での SpMV 計算はあまり多く検証されていないが, Nathan Bell ら [11] が NVIDIA GeForce GTX 280 を用いて検証しており，そこでは ELLPACK，CSR，COO 形式などと比較され，CDS 形式で単精度演算で 36GFLOPS，倍精度演算で 16GFLOPS となり，他の形式に比べて圧倒的に高いパフォーマンスを出していることが示されている．これまで CDS 形式があまり検証されていない理由として，フロリダ大コレクション [12] のような非ゼロ成分が完全にランダムな行列では CDS 形式による格納は膨大な記憶容量を必要となり，一般性に乏しいことからこれまであまり検証されていないと考えられる．

本研究で取り上げる有限要素法による離散化では優対角行列となり，列方向への大きさも有限要素節点により決定されるため特定の節点に過度に集中しなければ膨大な帯行列の行列とはならない [13]．

本研究で提案する手法は，最近の Kepler アーキテクチャをベースとする GPU を用いて，CDS の性質からなるループアンローリングの検証や Kepler アーキテクチャから導入された Read-Only データキャッシュ [14] の特性，また係数行列の対称行列を利用した省メモリを考慮した場合も検証する．

2. ポアソン方程式の離散化と行列の格納

2.1 ポアソン方程式の離散化

流体の運動方程式および連続の方程式において非圧縮性を考慮した場合，圧力に関するポアソン方程式が導かれる．この 3 次元ポアソン方程式を有限要素法により離散化を行う．

$$\frac{\partial^2 \phi}{\partial x_i^2} = f \quad in\ \Omega \tag{1}$$

任意の重み関数 w を用いて積分表現で表すと，次の重みつき残差方程式を得る．

$$\int_\Omega \left[\frac{\partial^2 \phi}{\partial x_i^2} - f \right] w d\Omega = 0 \tag{2}$$

ここで Γ は領域 Ω の境界とする．式 (2) の 2 階微分項に対して部分積分及びガウスの発散定理を用いると，次の弱形式を得る．

$$\int_\Omega \left[\frac{\partial w}{\partial x_i} \frac{\partial \phi}{\partial x_i} - wf \right] d\Omega = \int_\Gamma w \frac{\partial \phi}{\partial x_i} n_i d\Gamma \tag{3}$$

任意の重み関数 w および未知関数 ϕ に対して，領域 Ω 内を要素分割し，その要素内で次の補間関数を用いる．ここで重み関数および補間関数を同型で取るガラーキン有限要素法を適用する．

$$w = N_\alpha w_\alpha \tag{4}$$

$$\phi = N_\alpha \phi_\alpha \tag{5}$$

ここで α は各要素の要素節点番号を表す．式 (4)，(5) を代入すると，各要素の要素方程式を得る．

$$\int_{\Omega_e} \frac{\partial N_\alpha}{\partial x_i} \frac{\partial N_\beta}{\partial x_i} d\Omega \phi_\beta = \int_{\Gamma_e} N_\alpha \frac{\partial \phi}{\partial x_i} n_i d\Gamma - \int_{\Omega_e} N_\alpha f d\Omega \tag{6}$$

各要素内で得られた要素方程式を全体型への重ね合わせを行うことにより，次の連立方程式を得る．

$$\boldsymbol{K}\boldsymbol{\phi} = \boldsymbol{f} \tag{7}$$

このとき $\boldsymbol{K}$ は係数行列であり，$\boldsymbol{\phi}$ は未知関数（ベクトル），$\boldsymbol{f}$ は既知の右辺ベクトルとなる．

2.2 有限要素法による疎行列の性質

式 (7) の係数行列 $\boldsymbol{K}$ は疎な優対角な帯行列となる．さらにガラーキン有限要素法を用いた場合，式 (7) の行列は対称行列となる．この性質は質量行列，拡散行列にも同様であるが，上流化重み関数などを用いるとこの対称性が崩れる場合がある．有限要素法では帯行列の帯幅は各節点に関係する要素の数に依存し，一つの節点に多くの要素が関係する場合はその帯幅も大きくなる．このほか高次の補間関数を用いた場合も行列の帯幅は変化する．

2.3 疎行列の CDS 形式の格納例

帯行列の格納方法として，ベクトル計算機向けに開発された Compressed Diagonal Storage(CDS) 形式 [10] を採用する．その格納方法は，図 2 の行列では，図 3 のようになる．ここで図 3 に現れるアスタリスク記号 (*) は padding

される成分を表す．行列を記憶するときには必要であるが，計算からは除外される成分である．また対称性を考慮し，対角成分及び左側半分のみを記憶する場合は図 4 のような行列の記憶を行う．この方法では対称性を考慮しない場合に比べ，半分近くの記憶容量に削減できることになる．

このときの j は列の番号を示し，offset は演算においてゼロ成分を除くためのオフセット数を記憶する．列の数は要素形状や形状関数の次数に依存し，また節点の関係要素の数でも依存する．本計算で用いた要素分割法では $nd = 27$ となる．

$$\begin{pmatrix}
a_{00} & 0 & 0 & a_{03} & a_{04} & a_{05} & 0 & 0 & 0 \\
0 & a_{11} & 0 & 0 & a_{14} & a_{15} & a_{16} & 0 & 0 \\
0 & 0 & a_{22} & 0 & 0 & a_{25} & a_{26} & a_{27} & 0 \\
a_{30} & 0 & 0 & a_{33} & 0 & 0 & a_{36} & a_{37} & a_{38} \\
a_{40} & a_{41} & 0 & 0 & a_{44} & 0 & 0 & a_{47} & a_{48} \\
a_{50} & a_{51} & a_{52} & 0 & 0 & a_{55} & 0 & 0 & a_{58} \\
0 & a_{61} & a_{62} & a_{63} & 0 & 0 & a_{66} & 0 & 0 \\
0 & 0 & a_{72} & a_{73} & a_{74} & 0 & 0 & a_{77} & 0 \\
0 & 0 & 0 & a_{83} & a_{84} & a_{85} & 0 & 0 & a_{88}
\end{pmatrix}$$

図 1 帯行列

$$\begin{pmatrix}
a_{00} & & & & & & & & \\
0 & a_{11} & & & & & & & \\
0 & 0 & a_{22} & & & & symm. & & \\
a_{30} & 0 & 0 & a_{33} & & & & & \\
a_{40} & a_{41} & 0 & 0 & a_{44} & & & & \\
a_{50} & a_{51} & a_{52} & 0 & 0 & a_{55} & & & \\
0 & a_{61} & a_{62} & a_{63} & 0 & 0 & a_{66} & & \\
0 & 0 & a_{72} & a_{73} & a_{74} & 0 & 0 & a_{77} & \\
0 & 0 & 0 & a_{83} & a_{84} & a_{85} & 0 & 0 & a_{88}
\end{pmatrix}$$

図 2 対称帯行列

$$\overbrace{\begin{pmatrix}
* & * & * & a_{00} & a_{03} & a_{04} & a_{05} \\
* & * & * & a_{11} & a_{14} & a_{15} & a_{16} \\
* & * & * & a_{22} & a_{25} & a_{26} & a_{27} \\
* & * & a_{30} & a_{33} & a_{36} & a_{37} & a_{38} \\
* & a_{40} & a_{41} & a_{44} & a_{47} & a_{48} & * \\
a_{50} & a_{51} & a_{52} & a_{55} & a_{58} & * & * \\
a_{61} & a_{62} & a_{63} & a_{66} & * & * & * \\
a_{72} & a_{73} & a_{74} & a_{77} & * & * & * \\
a_{83} & a_{84} & a_{85} & a_{88} & * & * & *
\end{pmatrix}}^{nd}$$

図 3 Compressed Diagonal Storage(CDS)

$$\overbrace{\begin{pmatrix}
* & * & * & a_{00} \\
* & * & * & a_{11} \\
* & * & * & a_{22} \\
* & * & a_{30} & a_{33} \\
* & a_{40} & a_{41} & a_{44} \\
a_{50} & a_{51} & a_{52} & a_{55} \\
a_{61} & a_{62} & a_{63} & a_{66} \\
a_{72} & a_{73} & a_{74} & a_{77} \\
a_{83} & a_{84} & a_{85} & a_{88}
\end{pmatrix}}^{nd/2+1}$$

図 4 対称性を考慮した半要素記憶の CDS 形式

表 1 図 3 の行列に対するオフセット変数

j	0	1	2	3	4	5	6
offset	5	4	3	0	−3	−4	−5

表 2 図 4 の行列に対するオフセット変数

j	0	1	2	3
offset	5	4	3	0

表 3 NVIDIA GeForce TITAN 理論性能と CUDA

Compute capability	3.5
CUDA Cores	2688
L2 Cache Size[MB]	1536
Total global memory[MB]	6143
Memory Bandwidth[GB/s]	288.4
Peak SP TFLOPS (FMA*1)	4.5
Peak DP TFLOPS (FMA*1)	1.5
B/F, F/B (DP)	0.192, 5.20
CUDA	release 6.5

3. GPU による SpMV 計算

3.1 計算条件と計算環境

3 次元ポアソン方程式を双一次要素を用いたガラーキン型有限要素法により離散化し，得られた連立一次方程式に対して共役勾配法による求解を行う．計算条件として計算領域を正方領域とし，六面体要素による分割を行い，境界条件は適当なディリクレ条件（Dirichlet boundary condition）およびノイマン条件（Neumann boundary condition）を与える．収束条件として初期残差の 2-ノルムを 1.0^{-12} 以下とする．実数計算はすべて倍精度浮動小数演算を用いる．

表 3 に本研究で用いる GPU 計算環境を示す．

3.2 GPU による CDS 形式での SpMV プログラム

対称性を考慮しない場合の CUDA プログラムを図 5 に示す．このアルゴリズムについては，概ね文献 [11] の DIA sparse matrix format のプログラムと同等であるが，倍精

*1 fused multiply-add operation

```
__global__ void SpMV_cds_full(
    const    int  N,
    const double* A,
    const double* __restrict__ x,
          double* __restrict__ Ax,// ToDo
    const    int* __restrict__ offset,
    const    int  nd)
{
    unsigned int i = blockIdx.x * blockDim.x
                   + threadIdx.x;
    double tmp = 0.0;
    #pragma unroll 27 // ToDo
    for (int j = 0; j < nd; j++) {
        int ki = i + offset[j];
        if ( 0 <= ki && ki < N )
            tmp += A[j*N + i]   * x[ ki ];
    }
    Ax[i] = tmp;
}
```

図 5 CUDA による CDS 形式での SpMV

度浮動小数点演算であることや，ループアンローリングの利用，Read-Only データキャッシュの利用の点で異なる．また，文献 [11] での出力ベクトルにおいて加算代入が行われているが，これは単純代入で良い．

疎行列の CDS 形式では，対称性を考慮しないもの（図 5）と，対称性を考慮し左半分のみの半要素を記憶する方法で検証する．

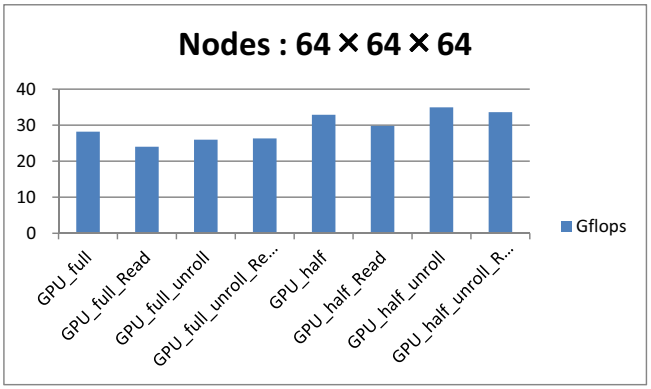
また図 5 での引数において，入力ベクトル及び offset 配列に対して Read-Only データキャッシュの使用をデフォルトとして明記するが，出力ベクトル（Ax[i]）に関しては入れた場合と入れない場合では計算速度において差異が見られたため，検証する．さらにこのアルゴリズムでは列方向にループアンローリングを行うことができるので，その有効性も比較を行い検証する．このときのループアンローリングの大きさは列の大きさとする．これら基本的な比較は，

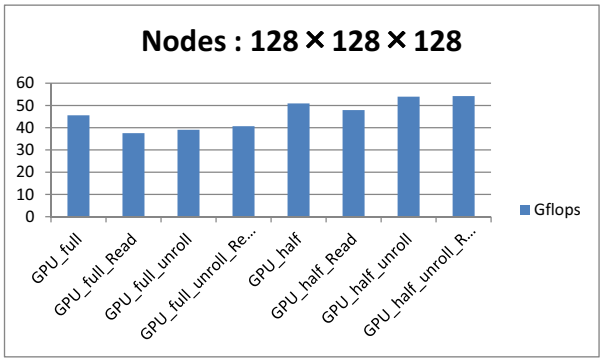
- 行列の対称性を考慮するか否か
- 出力ベクトルの Read-Only データキャッシュの明示の有無
- 列方向のループアンローリングの有無

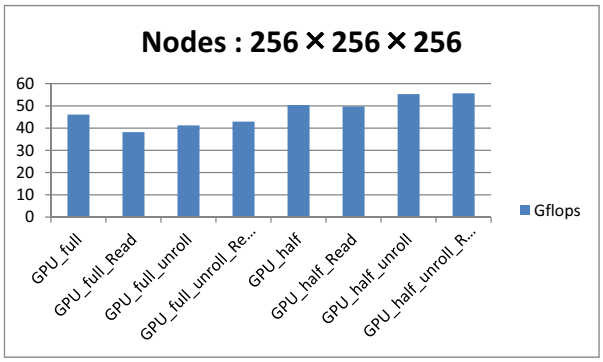
について検証する．これらの効果を検証するため，本研究で用いたケースを表 4 にまとめる．

3.3 計算結果と考察

演算性能の測定において，疎行列ベクトル積 (SpMV) のみを GPU で計算するものとし，CPU-GPU 間の通信時間は演算性能に含まれない．各ケース数回行い，最も演算性能が得られたものをそのデータとした．計算規模は要素分割において，各方向に $63 \times 63 \times 63$，$127 \times 127 \times 127$，







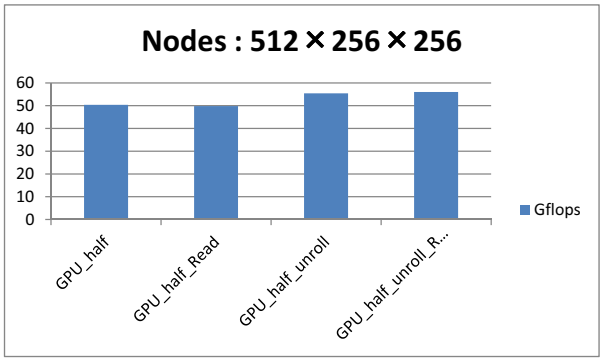


図 6 CDS 形式による SpMV の演算性能 (GFLOPS，倍精度計算)

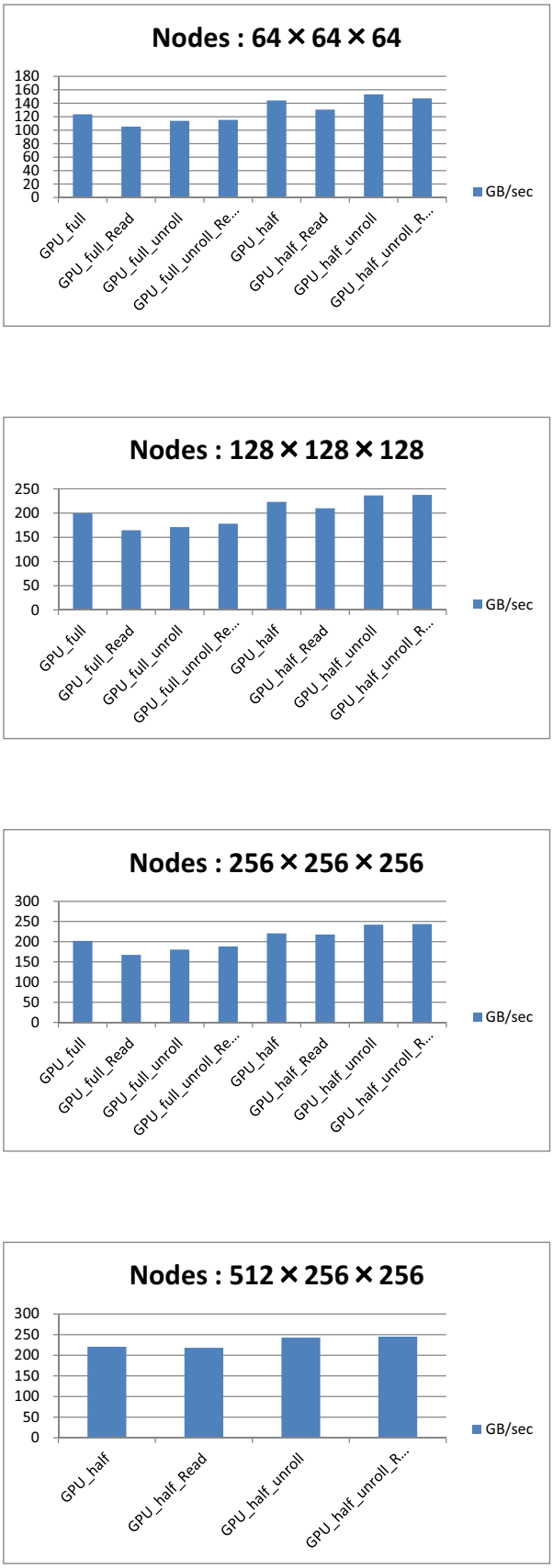


図 7 CDS 形式による SpMV のメモリバンド幅性能 (GB/s)

表 4 CDS 形式での GPU/CUDA 特性の検証

case	CDS の記憶	ループアンローリング	Read-only メモリ
GPU_full	full	-	-
GPU_full_Read	full	-	○
GPU_full_unroll	full	○	-
GPU_full_unroll_Read	full	○	○
GPU_half	half	-	-
GPU_half_Read	half	-	○
GPU_half_unroll	half	○	-
GPU_half_unroll_Read	half	○	○

$255 \times 255 \times 255$, $511 \times 255 \times 255$ の 4 パターンとし，このときの有限要素節点はそれぞれ，$64 \times 64 \times 64$, $128 \times 128 \times 128$, $256 \times 256 \times 256$, $512 \times 256 \times 256$ となり，これらの数値が総節点数 (N) となる．なお最大規模の計算 $512 \times 256 \times 256$ においては，対称性を考慮しない場合 GPU のメモリの上限を超えたため，計算不能となっている．また疎行列の CDS 化に対する padding 量は，$256 \times 256 \times 256$ において 0.26%である．なおこの割合は分割数を増やすごとに割合は小さくなっていく．

図 6 に個々の計算規模での，各ケースの演算性能 (GFLOPS) により比較した結果を示す．図 7 にメモリバンド幅の計測結果を示す．

FLOPS 値は，行列のゼロ成分を除いた nnz × 2 ×反復回数 (iter)/時間 [s] により算出している．

メモリバンド幅の計測は，行列から nnz × iter 回のロード，N の大きさの入力ベクトルが nnz × iter 回のロード，N の大きさの出力ベクトルへのストアが iter 回があるが，2 回目以降，入力ベクトルのデータはキャッシュにストアされているものとし，(nnz + 2N) × iter × sizeof(double)/時間 [s] により算出した．なおこの算出方法は後に示される CUDA プロファイラによる「Device Memory Read Throughput」と「Device Memory Write Throughput」の和 (238.5[GB/s]) で比較すると，GPU_half_unroll_Read での $256 \times 256 \times 256$ の場合で約 2.1%ほど高めの数値となっている．

対称性を考慮しない場合，ループアンローリング及び，出力ベクトルの Read-Only キャッシュの利用は，速度が低下する傾向が見られた．この場合で最も性能が高かったものは，節点数 $256 \times 256 \times 256$ の場合で，演算性能は 46.1GFLOPS，メモリバンド幅は 201.8[GB/s] であった．対称性を考慮しない場合での出力ベクトルの Read-Only データキャッシュの効果が得られなかった理由として，入力ベクトルでのキャッシュヒット率が低下したためだと考えられる．

一方，対称性を考慮し半成分のみの記憶で計算を行った場合，小さい規模での計算において出力ベクトルの Read-Only データキャッシュの指定をしない方が良い結果とな

る例外があるものの，ループアンローリングと出力ベクトルの Read-Only データキャッシュの利用に効果があることが分かった．この場合の性能が最も良かった節点数 $512 \times 256 \times 256$ の場合で，演算性能は 56GFLOPS，メモリバンド幅は 245.2[GB/s] の値を得た．この時のメモリバンド幅を理論性能値と比較すると，85% を達成したことになる．これは対称性に伴うループ内での演算回数が 2 倍となり，結果としてループアンローリングを行っていることと同じになったためと考えられる．さらに対称性を考慮した場合でのループアンローリングによる高速化の理由は，対称性を考慮しない場合に比べて，対称性による offset 配列のループアンローリングの展開が小さく，thread 当たりのレジスタの使用がそれほど大きくならなかったため，Achieved Occupancy の低下がそれほど大きくならずに演算が可能であったと考えられる．したがって演算性能は，行列の帯幅にも影響が大きいものと考えられる．

4. まとめ

ポアソン方程式の求解における疎行列ベクトル積 (SpMV) の計算に注目し，GPU を用いた CDS 形式による演算性能（FLOPS 値）及びデータ転送のメモリ帯域（GB/s）について検討を行った．ポアソン方程式の係数行列は対称性を考慮し，半要素を記憶しての計算を行った．これは記憶容量の半減を行うための左側行列を記憶したものを用いたが，右側行列を左側行列を用いるアルゴリズムにより結果的にループアンローリングを行っていることと同等になり，計算の向上を図ることが出来たと考えられる．また本手法ではオフセット配列，入力ベクトルで再帰的な利用のための Read-Only データキャッシュを利用することとし，さらに出力ベクトルにも効果を検討し，以下のことがわかった．

- 行列の対称性を考慮して行列を記憶する方法は，記憶容量の半減化と高速化に寄与する．
- 行列の対称性を考慮しない場合には列方向のループアンローリングは有効でないが，対称性を考慮した場合には非常に有効である．
- 行列の対称性を考慮しない場合には出力ベクトルでの Read-Only データキャッシュの明示は有効でないが，対称性を考慮した場合にはループアンローリングを併用することで高速化に有効である．

本研究での GPU 計算において，行列の対称性を考慮し最大性能値となった計算は最大規模であった $512 \times 256 \times 256$ での分割数のとき，演算性能は 56GFLOPS，メモリバンド幅は 245[GB/s] の性能となり，これは理論メモリバンド幅の 85%に達した．一方，対称性を考慮しない場合の最大性能値は，$256 \times 256 \times 256$ での分割数のときで，46[GFLOPS] であり，このときのメモリバンド幅は 202[GB/s] であった．

本計算では圧力ポアソン方程式の係数行列に対して対称行列となるため省メモリを考慮して検討したが，この対称性はガラーキン型有限要素法を用いての質量行列や拡散行列を陰的に扱った場合にも同様に対称行列となるため，同じ SpMV カーネルの計算を何ら変更することなく適用可能である．

参考文献

[1] 三浦慎一郎： 有限要素法によるチャネル乱流の LES 解析，東北大学 サイバーサイエンスセンター SENAC, Vol.40(2),pp.15-29 (2007)
[2] 棚橋隆彦：流れの有限要素法解析 I，朝倉書店，(1997)
[3] 藤野清次，張紹良：反復法の数理 ，朝倉書店 (1996)
[4] 椋木 大地，高橋 大介，GPU における高速な CRS 形式疎行列ベクトル積の実装, 情報処理学会研究報告，2013-HPC-138(5), pp.1-7(2013)
[5] NVIDIA : cuSPARSE, http://docs.nvidia.com/cuda/pdf/CUSPARSE_Library.pdf
[6] 田邊昇，冨森苑子，高田雅美，城和貴：疎行列ベクトル積性能を決める諸要因，情報処理学会研究報告，2014-HPC-143(7), 1-10(2014)
[7] John R. Rice, Ronald F. Boisvert : Solving Elliptic Problems Using ELLPACK, Springer Series in Computational Mathematics Volume 2(1985)
[8] Y. Saad : Krylov Subspace methods on Supercomputers, *Siam J. Sci. Stat. Comp.*, vol 10(6), pp. 1200-1232(1989).
[9] 長坂侑亮，額田彰，松岡聡：GPU のキャッシュを考慮した疎行列ベクトル積計算手法の性能評価，情報処理学会研究報告，2014-HPC-144(5), 1-9（2014）
[10] G. R. Brozolo, M. Vitaletti : Conjugate gradient subroutines for the IBM 3090 Vector Facility, *IBM J. DEVELOP.*, Vol.33, No. 2, pp.125-135(1989).
[11] Nathan Bell and Michael Garland : Efficient Sparse Matrix-Vector Multiplication on CUDA, NVIDIA Technical Report NVR-2008-004(December 2008).
[12] T. A. DAVIS, Y.HU : The University of Florida Sparse Matrix Collection, ACM Transactions on Mathematical Software, Vol 38, Issue 1, pp,1-25(2011)
[13] 谷口健男：FEM のための要素自動分割デローニー三角分割法の利用，森北出版，(1992)
[14] NVIDIA Corporation, NVIDIAGs Next-Gen CUDA Compute Architecture Kepler GK110, http://www.nvidia.com/content/PDF/kepler/NVIDIAKepler-GK110-Architecture-Whitepaper.pdf, (2012).

付　　録

A.1 GPU/CUDA for SpMV Performance with CDS format

A.2 nvprof

要素分割数 $255 \times 255 \times 255$，節点数 $256 \times 256 \times 256$ における，GPU_full_unroll_Read での nvprof の結果の一部を示す.

Multiprocessor Activity	99.97%
Achieved Occupancy	0.715628
Multiprocessor Activity	99.97%
Texture Cache Hit Rate	49.73%
Texture Cache Throughput	570.81GB/s
Device Memory Read Throughput	229.95GB/s
Device Memory Write Throughput	8.5374GB/s
Global Store Throughput	8.5374GB/s
Global Load Throughput	0.00000B/s
L2 Hit Rate (L1 Reads)	0.00%
L2 Hit Rate (Texture Reads)	42.87 %
L2 Throughput (Texture Reads)	402.51GB/s
Warp Execution Efficiency	100.00 %
Requested Non-Coherent Global Load Throu	515.31GB/s
L2 Throughput (Reads)	402.52GB/s
L2 Throughput (Writes)	8.5375GB/s
Warp Non-Predicated Execution Efficiency	98.92%
FP Instructions(Double)	451803646
Floating Point Operations(Double Preciso	435026430
L2 Throughput (Non-Coherent Reads)	402.51GB/s
L2 Non-Coherent Read Transactions	197745410
Non-Coherent Global Hit Rate	49.73%
Non-Coherent Global Memory Load Throughp	800.72GB/s
Non-Coherent Global Load Efficiency	64.36%
L2 Write Transactions (L1 write requests	4194304
L2 Transactions (Texture Reads)	197745022
L2 Throughput (L1 Writes)	8.5374GB/s

表 A·1 GPU による SpMV の計算性能（倍精度計算）

Elements	$63 \times 63 \times 63$		
Nodes	$64 \times 64 \times 64$		
N	262,144		
nnz	7,003,774		
Case	ms/iter.	GFLOPS	MB/s
GPU_full	0.488	28.20	123.54
GPU_full_Read	0.572	24.02	105.24
GPU_full_unroll	0.529	25.98	113.81
GPU_full_unroll_Read	0.523	26.30	115.25
GPU_half	0.418	32.88	144.06
GPU_half_Read	0.461	29.79	130.51
GPU_half_unroll	0.393	34.95	153.11
GPU_half_unroll_Read	0.409	33.59	147.17

Elements	$127 \times 127 \times 127$		
Nodes	$128 \times 128 \times 128$		
N	2,097,152		
nnz	56,327,422		
Case	ms/iter.	GFLOPS	MB/s
GPU_full	2.424	45.60	199.72
GPU_full_Read	2.943	37.56	164.51
GPU_full_unroll	2.829	39.08	171.15
GPU_full_unroll_Read	2.719	40.67	178.09
GPU_half	2.171	50.93	223.06
GPU_half_Read	2.308	47.91	209.82
GPU_half_unroll	2.048	53.97	236.38
GPU_half_unroll_Read	2.039	54.23	237.48

Elements	$255 \times 255 \times 255$		
Nodes	$256 \times 256 \times 256$		
N	16,777,216		
nnz	451,803,646		
Case	ms/iter.	GFLOPS	MB/s
GPU_full	19.24	46.10	201.82
GPU_full_Read	23.21	38.20	167.28
GPU_full_unroll	21.52	41.21	180.43
GPU_full_unroll_Read	20.65	42.94	187.99
GPU_half	17.60	50.39	220.62
GPU_half_Read	17.83	49.74	217.77
GPU_half_unroll	16.04	55.30	242.13
GPU_half_unroll_Read	15.94	55.65	243.65

Elements	$511 \times 255 \times 255$		
Nodes	$512 \times 256 \times 256$		
N	33,554,432		
nnz	903,607,294		
Case	ms/iter.	GFLOPS	MB/s
GPU_full	NA		
GPU_full_Read	NA		
GPU_full_unroll	NA		
GPU_full_unroll_Read	NA		
GPU_half	35.20	50.39	220.62
GPU_half_Read	35.61	49.81	218.10
GPU_half_unroll	31.99	55.44	242.72
GPU_half_unroll_Read	31.67	56.01	245.23