Vol.2018-HPC-164 No.3
2018/5/7

GPUにおけるSELL形式疎行列ベクトル積の実装と性能評価

佐藤 駿一[1,a)] 高橋 大介[2,b)]

概要：疎行列ベクトル積は科学技術計算で広く使われる重要な計算である．疎行列ベクトル積は疎行列の格納形式を，自身の構造と使用する計算機のアーキテクチャに応じて変更することにより，不必要な計算，記憶領域の使用を削減できる．そのため，これまでに様々な疎行列の格納形式が提案されてきた．近年アクセラレータとして GPU が注目されており，GPU に対応した疎行列の格納形式として SELL 形式が提案されている．GPU を使用し疎行列ベクトル積を行うライブラリとして，NVIDIA により cuSPARSE が提供されているが，このライブラリに SELL 形式は実装されていない．本研究では GPU が搭載されているシステムにおいて SELL 形式の疎行列ベクトル積の実装を行い，The University of Florida Sparse Matrix Collection から収集した疎行列を使用し，評価を行った．その結果，cuSPARSE と比較して最大で約 1.02 倍の性能向上を達成した．

1. はじめに

疎行列ベクトル積（Sparse Matrix Vector Multiplication）は，科学技術計算で広く使われる重要な計算である．疎行列ベクトル積を素朴に計算した場合，不要な計算と記憶領域の使用が発生する．疎行列は自身の構造と使用する計算機の特性を考慮して記憶領域に格納することにより，不要な記憶領域の使用を減らすことができる．また，疎行列ベクトル積を上記のように格納された疎行列に対して行うことにより不要な計算を減らし，高速化を行うことが可能である．そのため，これまでに様々な疎行列の格納形式が提案されてきたが，著者の知る限り，全ての計算機のアーキテクチャ，疎行列に対して実行性能が最も高くなる格納形式は存在しない．疎行列ベクトル積を高速に計算するには計算機のアーキテクチャ，疎行列毎の特性を考慮し，最適な格納形式の選択を行うことが重要である．

HPC 分野では計算の高速化を行うためにアクセラレータとして GPU（Graphics Processing Unit）が用いられる．GPU を汎用計算に用いる試みは GPGPU（General-Purpose computing on Graphics Processing Units）と呼ばれている．疎行列ベクトル積も同様に，GPU を用いた疎行列の格納形式と高速化手法が提案されている [1][2].

また，NVIDIA により疎行列計算に関するライブラリとして cuSPARSE[3] が提供されており，このライブラリ内の関数を利用することにより GPU を用いた疎行列計算の高速化を図ることが可能である．最新版の CUDA 9.1.85 に搭載されている cuSPARSE においてサポートされている疎行列格納形式は CRS（Compressed Row Storage）形式，BCRS（Block Compressed Row Storage）形式，HYB（HYBrid）形式，および ELL（ELLpack）形式である．ELL 形式は HYB 形式においてパラメータを変更することにより使用できる．しかし，cuSPARSE において SELL（Sliced-ELLpack）形式はサポートされていない．

本研究では GPU が搭載されているシステムにおいて SELL 形式を対象とした実装及び評価を行う，

2. 関連研究

疎行列の格納形式は様々なものが提案されており，COO（COOrdinate）形式，CRS 形式を基本とし，ELL 形式 [4]，SELL 形式 [2]，DIA（DIAgonal）形式 [5] 等の種類がある．

Xeon Phi や GPU など，メニーコアアーキテクチャに対する疎行列ベクトル積の高速化の研究も行われており，文献 [6] では疎行列格納形式として CRS 形式，ELL 形式が CPU，Xeon Phi，および GPU に対して実行されている．非零要素の少ない疎行列では，CPU の性能が高く，逆に非零要素の多い疎行列において Xeon Phi，GPU の性能が高い傾向が確認されている．

文献 [7] では疎行列格納形式として CRS 形式，ELL 形式に加え，SELL-P（padded sliced ELLPACK）形式が CPU

[1] 筑波大学大学院システム情報工学研究科
[2] 筑波大学計算科学研究センター
[a)] ssato@hpcs.cs.tsukuba.ac.jp
[b)] daisuke@cs.tsukuba.ac.jp

や GPU に対して実行されている．この研究における提案手法である SELL-P 形式の実行性能が cuSPARSE の実行性能に匹敵することが示されている．文献 [7] において提案された疎行列格納形式である SELL-P 形式では疎行列ベクトル積を行う際に複数のパラメタが出現する．

疎行列の格納形式におけるパラメタの概念の出現に従い，パラメタを自動でチューニングする需要が生じた．文献 [2] では GPU を使用して疎行列ベクトル積を計算する際に用いるスレッド数を自動でチューニングする方法が提案されている．

また，既存の格納形式を新しいアーキテクチャに適用する研究も行われており，文献 [8] では CRS 形式を GPU 上で高速に実行するために，GPU の機能を CRS 形式の疎行列ベクトル積に適用し，cuSPARSE に対して 200 種類の疎行列に対して平均して約 1.86 倍の性能向上を果たしている．

文献 [9] では疎行列を画像に変換し，この画像に対して機械学習を行い，疎行列を似た性質を持ったグループに分類する．これにより分類された疎行列のグループ毎に適する格納形式を提示することにより疎行列ベクトル積を高速化する手法が提案されている．

3. 疎行列ベクトル積

本章では疎行列ベクトル積について説明する．疎行列ベクトル積とは，疎行列とベクトルの積計算であり，M 行 N 列の疎行列 A，N 行のベクトル x，M 行のベクトル y を用いて式 (1) で表される．

$$y = Ax \tag{1}$$

疎行列とは構成要素の大部分が零要素の行列である．疎行列ベクトル積は疎行列を係数行列として用いる連立一次方程式，固有値問題，偏微分方程式などに用いられる．そのため，科学技術計算で広く用いられ，様々なアプリケーションにおいて重要な計算である．疎行列ベクトル積がアプリケーションの実行時間に及ぼす影響は非常に大きいものであるため，高速化が望まれている．

疎行列ベクトル積を素朴に計算した場合，疎行列の構成要素の大部分が零要素であるため，不必要な計算，記憶領域の使用が発生する．疎行列ベクトル積は疎行列の格納形式を疎行列の構造，また使用する計算機のアーキテクチャに適するように変更することにより，この不必要な計算，記憶領域の使用を削減できる．

しかし，疎行列ベクトル積は計算を行う計算機のアーキテクチャや計算に用いる疎行列に応じて最適な計算手法が異なる．そのため，これまでに様々な疎行列の格納形式が提案されてきた．

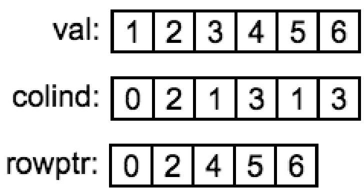

図 1　CRS 形式を用いた行列の格納例

```
function CRS(alpha, val, colind, rowptr, x, beta, y)
    for i ← 0 to N − 1 do
        for j ← rowptr[i] to rowptr[i + 1] − 1 do
            y[i] ← alpha × val[j] × x[colind[j]] + beta × y[i]
        end for
    end for
end function
```

図 2　CRS 形式疎行列ベクトル積の擬似コード

3.1 CRS 形式

CRS 形式は疎行列を各行毎に探索し，非零要素とその要素が格納されている列番号を用いて疎行列を表現する格納形式である．**図 1** に CRS 形式を用いて疎行列を格納した例を示す．上記のように格納することにより，非零要素のみを使用して疎行列ベクトル積を行うことが可能になる．CRS 形式は以下の 3 つの配列を用いる．

- val：非零要素の値を保持する配列
- $colind$：非零要素の列方向の添字を保持する配列
- $rowptr$：val，$colind$ において各行の先頭の要素が格納されている要素の添字を保持する配列

また，**図 2** に CRS 形式を用いた疎行列ベクトル積の擬似コードを示す．$alpha$，$beta$ は疎行列ベクトル積を行う際に各項に掛かる係数である．配列 x は行列 A に掛けるベクトルの値が格納された配列である．配列 y は疎行列ベクトル積の結果を格納する配列である．CRS は実装が比較的容易な反面，ベクトル x へアクセスする際に間接参照が行われる．そのため，疎行列ベクトル積を行う際にベクトル x へのアクセスが不連続になる可能性がある．また，並列化を試みる際に行単位での並列化を行うと，行毎の非零要素数の違いからロードバランスが悪くなる可能性がある．

4. SELL 形式疎行列ベクトル積の実装

本章では SELL 形式の説明，実装，そして高速化手法を説明する．

はじめに SELL 形式の説明を行う．SELL 形式は疎行列格納形式の 1 つである ELL 形式を改善したものである．ELL 形式は column major order で非零要素が格納され，内部ループのループ長が固定されるため，並列化した時のロードバランスが良い．しかし，疎行列を ELL 形式に変形し記憶領域に格納する際，行毎の非零要素数の違いによる列数の差異を無くすために，パディングを埋め込む必要がある．このパディングにより，疎行列ベクトル積を行う際

val: 1 3 2 4 5 6

colind: 0 1 2 3 1 3

slieced_start: 0 4 6

S: 2

図 3　SELL 形式を用いた行列の格納例

function SELL(S, alpha, val, clumn, sliced_start, x, beta, y)
　$s \leftarrow 0$
　while $s < N - 1$ **do**
　　for $r \leftarrow s$ to $s + S$ **do**
　　　$i \leftarrow sliced\_start[s/S] + r - s$
　　　while $i < sliced\_start[s/S + 1]$ **do**
　　　　$y[r] \leftarrow alpha \times x[column[i]] \times val[i] + beta \times y[r]$
　　　　$i \leftarrow i + S$
　　　end while
　　end for
　　$s \leftarrow s + S$
　end while
end function

図 4　SELL 形式疎行列ベクトル積の擬似コード

に本来不要な記憶領域の使用と計算が発生する．SELL 形式は ELL 形式において問題となる「パディングによる不要な記憶領域の使用と計算」を軽減させる格納方法である．SELL 形式は，任意のスライスサイズを決め，そのスライスサイズ毎に疎行列を行方向に分割する．その後に分割した行列毎に ELL 形式で行列を記憶領域に格納する．部分行列毎に ELL 形式で保存することにより，その行列の列数の基準となる，行毎の非零要素数を行列全体からではなく，部分行列毎に採用できる．そのため，部分行列毎に最適な非零要素数を選択でき，疎行列全体としてパディングを減らすことができる．**図 3** に SELL 形式を用いて疎行列を格納した例を示す．SELL 形式はスライスサイズ S と以下の 3 つの配列を用いて実装される．

- val：部分行列毎に非零要素の値を保持する配列
- $column$：部分行列毎の非零要素の列方向の添字の配列
- $sliced\_start$：$column$ において各部分行列の先頭が格納されている要素の添字の配列

次に SELL 形式を用いた疎行列ベクトル積の実装について説明する．**図 4** に擬似コードを，**図 5** に CUDA を使用したコードを示す．$alpha$，$beta$ は疎行列ベクトル積を行う際に各項に掛かる係数である．配列 x は行列 A に掛けるベクトルの値が格納された配列である．配列 y は疎行列ベクトル積の結果を格納する配列である．SELL 形式は ELL 行列をスライスサイズ S によって分割する形式であるが，スライスサイズ S はチューニングパラメタであるため，この値を調整することにより SELL 形式疎行列ベクトル積の性能が変化する．SELL 形式において $S = 1$ の場合，CRS

```
__global__ void SELLKernel
    (int M, int S, int numberOfSlicedMatrix,
     double alpha, double *value, int *column,
     double *x, double beta, double *y) {
    int s = blockDim.x * blockIdx.x;
    int r, slice_first, slice_end;
    double tval;

    while (s < M) {
        tval = 0.0;
        r = s + threadIdx.x;
        slice_first = tex1Dfetch(t_slice_start, s/S);
        slice_end = tex1Dfetch(t_slice_start, s/S+1);
        for (int i = slice_first + r - s;
                 i < slice_end; i+=S) {
            tval = alpha * x[column[i]] *
                   value[i] + beta * tval;
        }
        y[r] += tval;
        s += S * numberOfSlicedMatrix;
    }
}
```

図 5　CUDA を使用した SELL 形式疎行列ベクトル積のカーネルコード

形式と同様になる．また，$S = N$ の場合 ELL 形式と同様になる．

最後に本研究で使用した SELL 形式の高速化手法について説明する．本研究で使用した SELL 形式の高速化手法は「行毎の非零要素数を用いた行の並べ替え」と「sliced_start に対するテクスチャメモリの使用」の 2 つである．以降の小節において 2 つの高速化手法について説明する．

4.1　行毎の非零要素数を用いた行の並べ替え

行毎の非零要素数を用いた行の並び替えによる高速化は文献 [2] で提案されている手法である．SELL 形式はパラメタ S で分割サイズの指定を行い，疎行列を部分行列を分割し，部分行列毎に ELL 形式を行う格納形式である．しかし，単純に疎行列を部分行列に分割した場合，部分行列内の行列毎の非零要素数にばらつきが生じる．結果，部分行列に分割したとしても多数のパディングが必要となる可能性がある．反対に部分行列内の行毎の非零要素が近い場合，部分行列内のパディングの減少が期待できる．

そのため，部分行列を作成する前に部分行列内で各行毎の非零要素数が近い値になるように疎行列に対して行の並び替え操作を行う．部分行列内のパディングが並び替え操作を行う前に比べて減少することにより不要な計算を減らすことができるため，疎行列ベクトル積の高速化が期待できる．

4.2　sliced_start に対するテクスチャメモリの使用

GPU のスレッドの実行単位は warp と呼ばれ，GPU 内でカーネルが実行される際には 1 スレッド毎ではなく，warp 毎にカーネルが実行される．NVIDIA により開発された Pascal 世代と Volta 世代の GPU は 32 スレッドを 1 単位として扱う．SELL 形式の疎行列ベクトル積カーネルを GPU 上で実行する際に warp 内のスレッドは同じ部分

表 1　実験環境

プロセッサ名称・種別	NVIDIA Tesla P100	NVIDIA Tesla V100
コア数	3584	5120
動作周波数	1.328GHz	1.370GHz
倍精度浮動小数点理論演算性能	4.76TFLOPS	7.00TFLOPS
メモリ	16GB (HBM2)	16GB (HBM2)
メモリバンド幅	720GB/s	900GB/s
コンパイラ	nvcc（9.1.85）	nvcc (9.1.85)
コンパイルオプション	-arch=sm_60 -O3 -lcusparse	-arch=sm_70 -O3 -lcusparse
疎行列計算ライブラリ	cuSPARSE (cuda 9.1.85)	cuSPARSE (cuda 9.1.85)

表 2　評価に用いた疎行列

行列名	行数	列数	非零要素数
shyy41	4720	4720	20042
cz5108	5108	5108	51412
piston	2025	2025	100015
shipsec1	140874	140874	3568176
cant	62451	62451	4007383
PWTK	217918	217918	11524432
ldoor	952203	952203	42493817
Hook_1498	1498023	1498023	59374451
audikw_1	943695	943695	77651847
ML_Geer	1504002	1504002	110879972

行列について疎行列ベクトル積を行っている．

SELL 形式において格納形式内で各部分行列毎の境界を示す値を保持する sliced_start について，上記の理由から warp 内のスレッドは疎行列ベクトル積を行う際のメモリアクセスには空間的局所性があることが考えられる．そのため，sliced_start の格納について，GPU 上でグローバルメモリの代わりにテクスチャメモリを使用する．テクスチャメモリを使用することにより，sliced_start へのメモリアクセスを各スレッド毎に個別に行うのではなく，warp を更に半分にしたグループ毎に一度に行うようにした．これにより，メモリアクセスの回数を減少でき，疎行列ベクトル積の高速化が期待できる．

5.　性能評価

本研究では NVIDIA の Pascal 世代と Volta 世代の GPU に対して CRS 形式と SELL 形式を実装し，NVIDIA により提供されている疎行列計算ライブラリである cuSPARSE を用いた実装を比較対象として性能評価を行った．Pascal 世代の GPU として NVIDIA Tesla P100 を，Volta 世代の GPU として NVIDIA Tesla V100 を使用した．本研究では疎行列ベクトル積の計算部分のみを実行時間計測の対象とした．また，疎行列ベクトル積は実行時間が非常に短く，計測を行う度に計測結果に揺らぎが生じるため，最初の 1 回を除き，10 回計測を行い，その平均値を実行時間として用いた．疎行列の非零要素のデータ型は倍精度浮動小数点である．本研究で評価に使用した疎行列は The University of Florida Sparse Matrix Collection[10] に登録されている疎行列を使用した．実験で使用した GPU を**表 1** に示す．使用した疎行列の一覧を**表 2** に示す．

本研究では以下の 3 つの性能評価を行った．

- Tesla V100 における我々が実装した疎行列ベクトル積と cuSPARSE を用いた疎行列ベクトル積の性能評価
- Tesla V100 における SELL 形式疎行列ベクトル積に含まれるパラメタ S の変化に対する性能評価
- Tesla P100 と Tesla V100 における我々が実装した疎行列ベクトル積の性能評価

以降の図において縦軸は実行性能を，横軸は性能評価に用いた行列を示す．また，行列は非零要素数が昇順となるように並べられている．

Tesla V100 における我々が実装した疎行列ベクトル積と cuSPARSE を用いた疎行列ベクトル積の性能評価の結果を**図 6** に示す．疎行列の非零要素数に実行性能が比例する傾向があることが見受けられた．我々の実装について注目すると，7 種類の行列について SELL 形式が CRS 形式に比べて高い性能を発揮した．一方，疎行列 shipsec1，PWTK，audikw_1 について CRS 式が SELL 形式に比べて性能が高い．疎行列の性質に応じて同じアーキテクチャを使用しても最適な格納形式が異なることが確認された．cuSPARSE を用いた実装において各行列毎に最も性能の高い格納形式と SELL 形式を比べた場合，我々が実装した SELL 形式は cuSPARSE に比べて平均で約 78%の性能となった．疎行列 piston において cuSPARSE に比べて最小の約 38%の性能であった．疎行列 ldoor において cuSPARSE に比べて最大の約 102%の性能を達成した．

次に Tesla V100 における SELL 形式実装に含まれるパラメタ S の変化に対する場合の性能評価の結果を**図 7** に示す．全体を通してパラメタ $S = 32$ の場合の性能が高い．GPU のスレッド実行単位である warp が 32 スレッドを 1 単位としているため，無駄なくスレッドを使用して計算を行うことができているからであると考えられる．スライスサイズが 32 より小さい場合，warp 内で異なる部分行列に対して疎行列ベクトル積の計算を行う必要があり，異なる sliced_start の値を読みに行く必要があるため，テクスチャメモリの使用による高速化が機能していない可能性がある．疎行列 cant や ML_Geer において $S = 1024$ の場合の

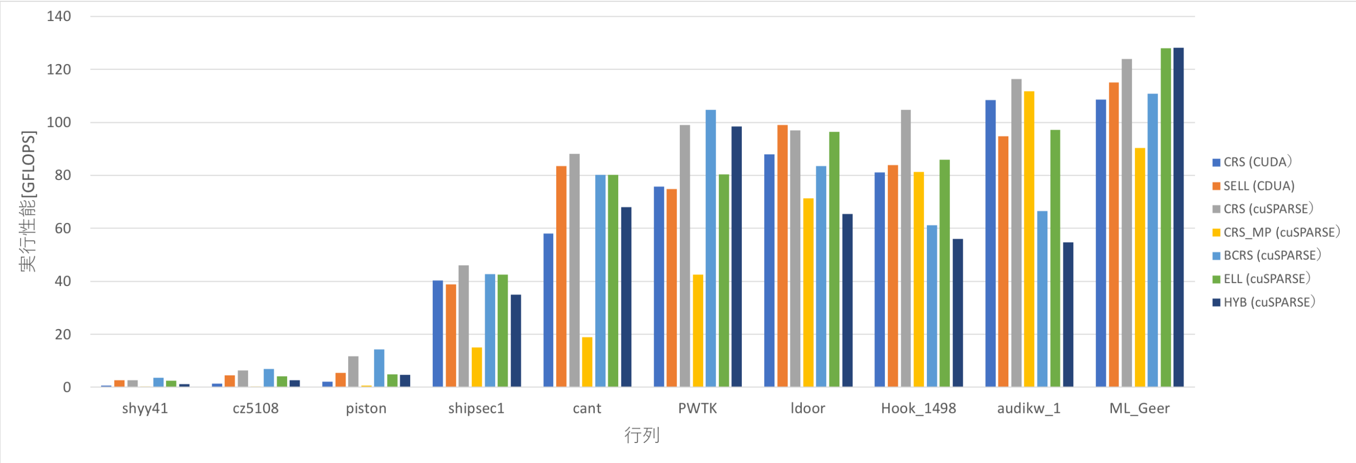

図 6 我々が実装した CRS 形式，SELL 形式の疎行列ベクトル積と cuSPARSE 9.1.85 を用いた疎行列ベクトル積の性能評価（Tesla V100）

図 7 我々が実装した SELL 形式の疎行列ベクトル積のパラメタ S の変化に対する性能評価（Tesla V100）

方が性能が高いなど行列によって最適なパラメタ S の値は異なる．そのため，Tesla V100 においても SELL 形式を実装する際にはパラメタ S のチューニングが必要であることが確認された．

最後に Tesla P100 と Tesla V100 における我々が実装した疎行列ベクトル積の性能評価の結果を**図 8** に示す．CRS 形式，SELL 形式全体を通して Tesla V100 で実行した疎行列ベクトル積は Tesla P100 で実行した場合に比べて実行性能の向上が見受けられる．これは疎行列ベクトル積がメモリアクセスに律速される計算であり，そして Tesla V100 が Tesla P100 に比べてメモリバンド幅が増加しているため，実行性能が向上したと考えられる．また，疎行列 PWTK において，Tesla P100 では SELL 形式が最適な格納形式であったが，Tesla V100 においては CRS 形式が SELL 形式に比べて性能が高い．アーキテクチャが異なることにより最適な格納形式が変わることが確認された．

6. まとめ

本研究では Pascal 世代と Volta 世代の GPU を用い，CRS 形式，SELL 形式の実装及び cuSPARSE との性能評価を行った．SELL 形式について行毎の非零要素数に応じて行を入れ替えること，また SELL 形式で疎行列を部分行列に分割する際の分割サイズの指定に使用される変数 S に対しグローバルメモリの代わりにテクスチャメモリにデータを格納することによりメモリアクセスを効率化することを試みた．結果として cuSPARSE に対して平均で約 78%の性能を，最大で約 102%の性能を実現した．

次に，SELL 形式で疎行列を部分行列に分割する際の分割サイズの指定に使用されるパラメタ S について，Volta 世代 GPU においても行列毎に最適な値は異なるため，チューニングの余地があることが確認された．

最後に，Pascal 世代と Volta 世代の GPU のアーキテクチャの違いにより，同じ疎行列に対して異なる格納形式が適する場合があることを確認した．

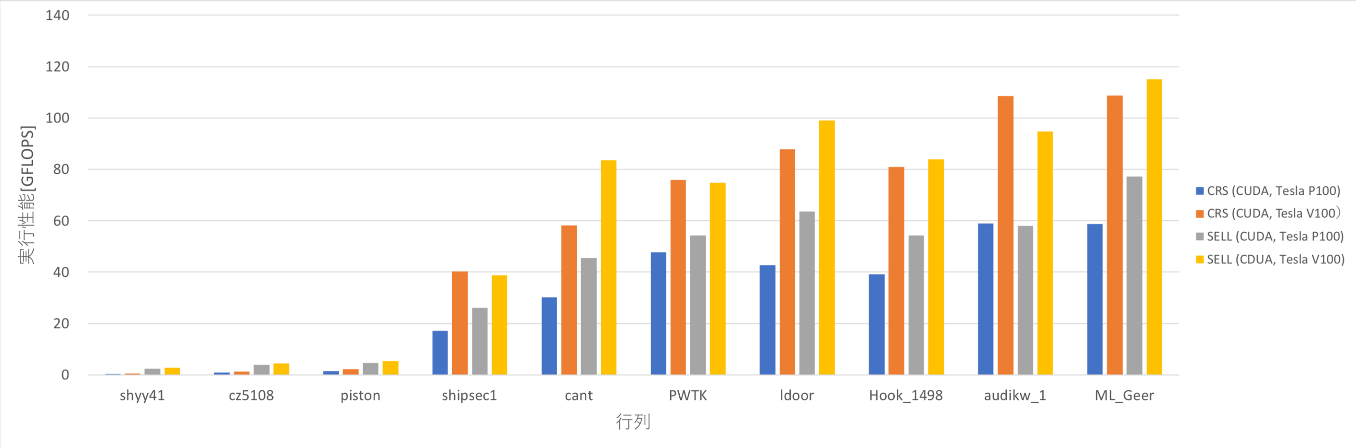

図 8 我々が実装した CRS 形式，SELL 形式の疎行列ベクトル積に対する異なる世代の GPU 間の性能評価（Tesla P100，Tesla V100）

参考文献

[1] Bell, N. and Garland, M.: Efficient sparse matrix-vector multiplication on CUDA, Technical report, Nvidia Technical Report NVR-2008-004, Nvidia Corporation (2008).

[2] Monakov, A., Lokhmotov, A. and Avetisyan, A.: Automatically tuning sparse matrix-vector multiplication for GPU architectures, *Proc. 5th International Conference on High Performance and Embedded Architectures and Compilers (HiPEAC 2010)*, Springer, pp. 111–125 (2010).

[3] NVIDIA: cuSPARSE Library (included in CUDA Tool kit), https://developer.nvidia.com/cusparse.

[4] Rice, J. R. and Boisvert, R. F.: *Solving elliptic problems using ELLPACK*, Vol. 2, Springer Science & Business Media (2012).

[5] Saad, Y.: *Iterative methods for sparse linear systems*, p. 520, Society for Industrial and Applied Mathematics (2003).

[6] 大島聡史，金子勇，片桐孝洋：Xeon Phi における SpMV の性能評価，情報処理学会研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 140, No. 33 (2013).

[7] Anzt, H., Tomov, S. and Dongarra, J.: Implementing a Sparse Matrix Vector Product for the SELL-C/SELL-C-σ formats on NVIDIA GPUs, Technical report, University of Tennessee, Tech. Rep. ut-eecs-14-727 (2014).

[8] 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ハイパフォーマンスコンピューティング (HPC)， Vol. 2013, No. 5, pp. 1–7 (2013).

[9] 滝沢寛之，崔航，平澤将一：機械学習によるコード最適化の可能性，計算工学講演会論文集，Vol. 22 (2017).

[10] Tim, D., Yifan, H. and Scott, K.: University of Florida Matrix Collection, https://sparse.tamu.edu/.