疎行列のステンシル構造の活用による疎行列ベクトル積の性能向上の調査

深谷 猛 [1)], 三浦 瑛絵 [2)], 岩下 武史 [1)]

1) 北海道大学 情報基盤センター

2) 北海道大学 工学部

fukaya@iic.hokudai.ac.jp

An investigation into the performance improvement of sparse-matrix vector multiplication by exploiting the stencil structure in a sparse matrix

Takeshi Fukaya[1)], Akie Miura[2)], Takeshi Iwashita[1)]

1) Information Initiative Center, Hokkaido University

2) School of Engineering, Hokkaido University

概要

疎行列ベクトル積は科学技術計算における重要な計算カーネルの一種である。シミュレーションで現れる疎行列の疎構造は多種多様であるが，一方で，直交格子を用いた差分解析等で現れる疎行列は，格子点同士の依存関係（ステンシル）に由来する特徴的な疎構造を有する。このような疎行列に対して，汎用的な疎行列ベクトル積の実装を行うよりも，特徴的な構造を考慮した実装を行う方が，性能面で有利となることが期待される。本論文では，ケーススタディを通して両者の性能比較を行い，構造を考慮することで得られる性能向上を定量的に調査する。

1 はじめに

疎行列とベクトルの積の計算（疎行列ベクトル積）は工学や自然科学の分野における様々な計算で頻出する計算カーネルの一種である。例えば，疎行列を係数とする大規模連立一次方程式の代表的な反復解法であるクリロフ部分空間法では，疎行列ベクトル積の繰り返しによりクリロフ部分空間を生成・拡大する。多くの場合，疎行列ベクトル積の計算時間が全体の計算時間の大部分を占めることが多く，そのため，高性能計算の分野では疎行列ベクトル積の高速化に関する様々な研究が行われている [1, 2]。

疎行列は名前の通り，行列の要素の大部分がゼロである行列である。疎行列における非ゼロ要素の配置（疎構造）は，疎行列の由来によって多種多様であるが，特定の解析では特徴的な疎構造となることが知られている。例えば，直交格子を用いた差分解析では，格子点同士の依存関係（通称，ステンシル）に由来する疎構造（以下，ステンシル構造）が現れる。

このように，特徴的な疎構造を疎行列が有する場合，その構造を考慮して疎行列ベクトル積の実装を工夫することで，汎用的な実装に対して性能が向上することが期待できる。現在，我々は，上記のステンシル構造に着目し，これを自動的に活用して疎行列ベクトル積の性能を改善する手法を研究している [3, 4]。本論文では，上記の研究の予備評価として行った，ステンシル構造の活用による疎行列ベクトル積の性能向上に関する調査結果を報告する。具体的には，3 次元空間における 7 点差分解析で現れる疎構造を模倣したテスト行列について，汎用的な疎行列ベクトル積の実装と，ステンシル構造を活用した実装の性能を二種類の計算機環境で比較する。実験結果から，ステンシル構造を活用する利点を確認するとともに，現状の計算機環境において得られる性能向上を定量的に評価する。

2 疎行列とステンシル構造

ステンシル構造を有する疎行列について，具体例を用いて説明する。3 次元のラプラス方程式 $\nabla^2\phi(x,y,z) = 0$ を，直交格子で空間を離散化し，偏微分を中心差分で近似すると，ある格子点と近隣の格子点との間に 7 点ステンシルと呼ばれる関係が生じる。そして，各格子点を辞書式順序付けで並べてベクトルにすると，依存関係を表す行列は図 1 に示したような特徴的な非ゼロ要素の配置を持つ。図 1 から明らかなように，非ゼロ要素は行列の特定の斜めのライン上のみに存在している。以降，本論文では，このような非ゼロ要素の配置を持つ疎行列のことを「ステンシル構造」を持つ疎行列を呼ぶ。

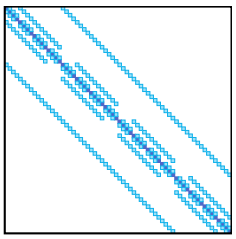

図 1: 7 点ステンシルに由来する疎行列

$$\begin{pmatrix} 1 & 0 & 2 & 3 \\ 4 & 5 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 7 & 8 \end{pmatrix}$$

図 2: 疎行列の例

val	1	2	3	4	5	6	7	8
col_ind	1	3	4	1	2	3	3	4

row_ptr	1	4	6	7	9

図 3: CRS 形式による格納の例

```
#pragma omp parallel for
for(i=0;i<n;i++){
  j = row_ptr[i];
  while(j<row_ptr[i+1]){
    y[i] += val[j] * x[col_ind[j]];
    j++;
}}
```

図 4: CRS 形式を用いた疎行列ベクトル積の実装

$$\begin{pmatrix} 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 3 & 0 & 0 & 4 & 0 \\ 5 & 0 & 6 & 0 & 0 & 7 \\ 0 & 8 & 0 & 9 & 0 & 0 \\ 0 & 0 & 10 & 0 & 11 & 0 \\ 0 & 0 & 0 & 12 & 0 & 13 \end{pmatrix}$$

図 5: Diagonal 形式が想定する疎構造

3 疎行列ベクトル積の実装の概要

本節では，汎用的な疎行列の格納形式である CRS 形式とそれに基づく疎行列ベクトル積の実装，および，ステンシル構造に適した Diagonal 形式とそれに基づく疎行列ベクトル積の実装の概要を述べる。

3.1 CRS 形式

CRS（Compressed Row Storage）形式 [5] は幅広く用いられる疎行列の格納形式の一種であり，多くの疎行列向け数値計算ライブラリでサポートされている。CRS 形式は，疎行列の非ゼロ要素のパターン（疎構造）を仮定しておらず，任意の疎行列を扱うことができる汎用性の高い格納形式である。

CRS 形式では，疎行列の非ゼロ要素を行ごとにまとめ，要素の値を配列 val，列のインデックスを配列 col_ind に格納し，両者の配列において，各行列の最初の要素の位置を配列 row_ptr に格納する．例えば，図 2 に示した行列を CRS で格納した場合，図 3 のようになる。

次に，CRS 形式を用いた場合の一般的な疎行列ベクトル積の実装を図 4 に示す。図 4 から分かるように，CRS 形式を用いた場合には，疎行列の非ゼロ要素の位置に関する仮定がないため，列のインデックスを配列 col_ind から取得した上で，対応する入力ベクトル $\boldsymbol{x}$ の要素を参照することになる。これは，配列 x の間接参照と呼ばれ，SIMD 化等のコンパイラによる最適化が施しにくくなるなど，高性能計算の観点からは好ましくないことが知られている。

3.2 Diagonal 形式

Diagonal 形式 [6] は，疎行列が特殊な構造，具体的には，2 節で示したような，特定の斜めのライン上のみに非ゼロ要素が存在する構造を持つ場合に限定した格納形式である。Diagonal 形式では，非ゼロ要素が存在する斜めのラインをそれぞれまとめて，配列 val に左上の要素から連続に要素の値を格納する。また，それぞれの斜めのラインと対角との距離の情報を配列 col_offset に格納する。例えば，図 5 のような疎行列を Diagonal 形式で格納すると，図 6 のようになる。

Diagonal 形式を用いた場合の疎行列ベクトル積の実装の例を図 7 に示す。ここで，K は斜めのラインの個数（col_offset の要素数）である．また，図 7 の実装例では，i に関するループの始まりと終わりを同じにするために，配列 x のサイズを拡大している．図 7 から分かるように，Diagonal 形式を用いた場合，疎行列ベクトル積の実装において，間接参照を回避することができる。また，CRS 形式を用いた場合と比べて，コンパイラによる最適化の効果がより期待できる。

一方，Diagonal 形式を用いた場合に懸念される点として，最内のループにおいて，結果を格納する配列 y が再利用されない点が挙げられる。これは，行列のサイズがキャッシュメモリのサイズよりも大きい場合に，k のループの各々で，配列 y のデータをメインメモリから参照（へ書き込み）する必要があることを意味している。この点に対しては，図 8 のように，タイリングを施すことで対応が可能である。図 8 では，タイリングのサイズを NB を適切に設定することで，k に関するループにおいて，配列 y の該当部分がキャッシュメモリ上に存在し続けることが期待される。なお，図 8 では，タイリングに伴う端数の処理に関する記述は省略している。

val[1][ ]	0	0	5	8	10	12
val[2][ ]	1	3	6	9	11	13
val[3][ ]	2	4	7	0	0	0

col_offset	-2	0	3

図 6: Diagonal 形式による格納の例

```
#pragma omp parallel
for(k=0;k<K;k++){
  j_off = col_offset[k];
  #pragma omp for
  for(i=0;i<n;i++){
    y[i] += val[k][i] * x[i+j_off+(n-1)];
}}
```

図 7: Diagonal 形式を用いた疎行列ベクトル積の実装

```
#pragma omp parallel for
for(ii = 0;ii<n;ii+=NB){
  for(k=0;k<K;k++){
    j_off = col_offset[k];
    for(i=ii;i<NB;i++){
      y[i] += val[k][i] * x[i+j_off+(n-1)];
}}}
```

図 8: Diagonal 形式を用いた疎行列ベクトル積の実装（タイリング版）

4 実験

4.1 実験内容

今回の実験では，7 点ステンシル構造を模倣した疎行列として，

$$a_{ij} = \begin{cases} -1 & (i = j \pm (nx)^2,\ j \pm nx,\ j \pm 1) \\ 6 & (i = j) \\ 0 & otherwise \end{cases}$$

を考える。ここで，行列 A の i 行 j 列の成分を $a_{i,j}$ としている。また，nx は空間の各次元の格子点数（今回は全ての次元で同じ）に相当し，行列のサイズは $n := (nx)^3$ となる。そして，この行列 A と n 次元ベクトル $\boldsymbol{x}$ の積の計算 $\boldsymbol{y} := A\boldsymbol{x}$ の計算時間を評価する。

4.2 計算機環境

今回の実験では，北海道大学情報基盤センターの SR16000 の 1 ノードと京都大学術情報メディアセンターの Laurel 2（システム B）の 1 ノードを利用した。それぞれのシステムの構成は表 1 に示した通りである。なお，コンパイルオプションは，SR1600 では `-q64 -O3 -qsmp=omp -qarch=pwr7 -qtune=pwr7 -qmaxmem=-1`，Laurel 2 では `-mcmodel=medium -shared-intel -qopenmp -O3 -xHost -ipo` を用いた。

表 1: 実験に使用した計算機環境

システム	SR16000	Laurel 2
プロセッサ	POWER7	Intel Xeon E5-2695v4
周波数	3.83GHz	2.10GHz
コア数/プロセッサ	8	18
プロセッサ数/ノード	4	2
ピーク演算性能/ノード	980GFLOPS	605GFLOPS
コンパイラ	IBM XLC	Intel icc

4.3 結果

行列サイズ $n = nx^3$ とし，nx を変えて，疎行列ベクトル積を 100 回繰り返した際の実行時間を測定した。なお，行列やベクトルの初期化の時間は測定対象外とした。実験結果を表 2(a) および (b) に示す。ここで，CRS は図 4，Diag は図 7，Diag(NB) は図 8 に示した内容に基づく実装である。

CRS と Diag（タイリング版を含む）を比較すると，タイリングのサイズを適切に設定した場合，CRS よりタイリング版の Diag 形式を用いた実装の方が，どちらの環境でも高速となっていることが確認できる。また，タイリングをしていない Diag 形式では CRS の場合よりも実行時間が増加していることも確認できる。このことから，今回実験を行った環境では，Diag 形式により間接参照を回避することよりも，結果のベクトルの再利用性が低下する影響の方が大きかったと推測できる。また，OpenMP によるスレッド並列化において，CRS 形式のほうがより外側のループを並列化できるという点が有利になった可能性も考えられる。この点を Diag 形式にタイリングを施すことで解消した場合，間接参照を回避することのメリットにより，CRS よりも高速となったと考えるのが自然である。

5 おわりに

本論文では，疎行列ベクトル積の計算に関して，行列が 7 点ステンシルに由来する特徴的な疎構造を有している場合に，汎用の格納形式である CRS 形式を用いた実装と，構造を考慮した Diagonal 形式（とその変種）を用いた実装とで，どの程度の性能差が生じるか検証を行った。今回実験を行った環境のどちらにおいても，適切なタイリングサイズでタイリングを施した Diagonal 形式に基づく実装が，CRS 形式に基づく実装よりも，1.3 倍から 1.5 倍程度性能が高いことが確

表 2: 疎行列ベクトル積 100 回の実行時間（秒）．括弧内は CRS 形式を用いた場合に対する高速化率．

(a) SR16000（32 スレッド使用）

nx	CRS	Diag	Diag [64]	Diag [128]	Diag [256]	Diag [512]	Diag [1024]
100	0.08	0.42 (0.19)	0.05 (1.53)	0.05 (1.53)	0.05 (1.56)	0.05 (1.59)	0.05 (1.52)
200	1.45	2.39 (0.61)	0.94 (1.55)	0.94 (1.54)	0.91 (1.60)	0.92 (1.58)	0.94 (1.54)
300	4.62	7.68 (0.60)	3.47 (1.33)	3.38 (1.37)	3.40 (1.36)	3.36 (1.37)	3.34 (1.38)
400	12.0	17.4 (0.69)	8.76 (1.38)	8.74 (1.38)	8.62 (1.40)	8.57 (1.41)	8.57 (1.41)
500	20.7	33.9 (0.61)	15.5 (1.34)	15.4 (1.35)	15.3 (1.35)	15.3 (1.36)	15.3 (1.35)

(b) Laurel 2（36 スレッド使用）

nx	CRS	Diag	Diag [64]	Diag [128]	Diag [256]	Diag [512]	Diag [1024]
100	0.05	0.03 (1.77)	0.30 (0.17)	0.17 (0.29)	0.09 (0.53)	0.05 (0.97)	0.03 (1.79)
200	1.21	1.70 (0.71)	2.78 (0.43)	1.64 (0.74)	0.95 (1.27)	0.96 (1.26)	0.84 (1.44)
300	3.22	5.12 (0.63)	8.95 (0.36)	5.37 (0.60)	3.11 (1.04)	2.49 (1.29)	2.35 (1.37)
400	7.57	12.3 (0.62)	21.0 (0.36)	12.6 (0.60)	7.16 (1.06)	5.95 (1.27)	6.10 (1.24)
500	15.9	25.3 (0.63)	40.6 (0.39)	23.5 (0.68)	15.0 (1.06)	11.6 (1.38)	11.9 (1.34)

認された。このことから，ライブラリを開発する場合に，ステンシル構造のように，それなりの頻度で出現する非ゼロ構造に特化した格納形式や実装をサポートすることで，性能面で利点があることが示された。

本論文で報告した内容を踏まえて，現在，我々は入力された疎行列に対して，Diagonal 形式と CRS 形式を組み合わせた格納形式を自動的に適用する仕組みを構築している [3, 4]。この仕組みをライブラリ等に組み込むことで，ユーザの負担を増加させることなく，特定の構造を行列が有していた場合に，その構造を生かした実装を自動的に適用することで性能を向上させることが可能となる。

■**謝辞**: 疎行列ベクトル積の実装に関して有益なご議論をいただいた京都大学学術情報メディアセンターの中島浩教授に感謝いたします．本研究の一部は，JSPS 科研費 JP15K16000 の助成を受けています．

参考文献

[1] C. Liu, M. Smelyanskiy, E. Chow, and P. Dubey, Efficient Sparse Matrix-vector Multiplication on x86-based Many-core Processors, Proceedings of the 27th International ACM Conference on International Conference on Supercomputing (ICS'13), pp. 273–282, 2013.

[2] M. Kreutzer, G. Hager, G. Wellein, H. Fehske, and A. Bishop, A Unified Sparse Matrix Data Format for Efficient General Sparse Matrix-Vector Multiplication on Modern Processors with Wide SIMD Units, SIAM J. Sci. Comput., Vol. 36, No. 5, pp. C401–C423, 2014.

[3] 深谷 猛, 三浦 瑛絵, 岩下 武史, ステンシル構造を利用した疎行列ベクトル積の高速化, 計算工学講演会論文集, Vol. 22, pp. 1–4, 2017.

[4] 石田 幸輝, 三浦 瑛絵, 深谷 猛, 岩下 武史, 中島 浩, 複数の格納形式を利用した疎行列ベクトル積の高速化に関する検討, 2017 年ハイパフォーマンスコンピューティングと計算科学シンポジウム（HPCS2017）, 2017.

[5] R. Barrett, W. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM, 1994.

[6] Y. Saad, Iterative Methods for Sparse Linear Systems, SIAM, 2003.