

複数アーキテクチャ上での疎行列ベクトル積の性能最適化手法

西 田 晃^{†1}

本稿では、現在開発を進めている並列反復解法ライブラリ Lis の複数アーキテクチャ上での性能最適化手法について検討する。反復解法の性能は、行列の形状、並列化手法、計算機のメモリ階層等によって大きく性能が変化する。ここでは、反復解法の要素演算である疎行列ベクトル積の性能を事前に評価することにより、反復解法の性能を最適化するための手法の妥当性について考察する。

Performance Optimization of Sparse Matrix-Vector Product on Multiple Architectures

AKIRA NISHIDA ^{†1}

This study discusses the performance optimization of the the parallel iterative solver library Lis, which is developed in our project. The performance of iterative solvers is affected by the data structure of matrices, the methodology of their parallelization, the memory hierarchy of the computers etc. In this paper, we evaluate the validity of the performance optimization of iterative solvers using the preliminary performance evaluation of sparse matrix vector products.

1. 背 景

本研究では、平成 14-19 年度科学技術振興機構 CREST 事業の一環として、反復解法ライブラリ Lis ^{*1} を開発、配布し、様々な並列計算機上で大規模な線形方程式を解くための環境

を提供している。また平成 20 年度には九州大学情報基盤研究開発センターにおいて固有値解法の実装を行い、同年 11 月より疎行列固有値解法に対応した新版を公開している。

本稿では、既に反復解法ライブラリ Lis に実装している疎行列ベクトル積の性能ベンチマークプログラムを発展させ、多様なアーキテクチャ上で評価を行った。

2. Lis を用いた実装

本研究では、以前より開発を進めている Lis 上に、Krylov 部分空間法を中心とする多様な反復解法を実装している。同様なライブラリとして、Argonne 米国立研究所の並列反復解法ライブラリ PETSc や Lawrence Berkeley 米国立研究所による並列反復解法ライブラリ Hypra などが知られている¹⁾。これらのライブラリでは、いずれもオブジェクト指向に基づいた設計を行っており、並列化はライブラリのレベルで実現されている。また、すべてのデータは API を用いて処理されている。すなわち、行列、ベクトルデータ等の生成・廃棄、及びこれらのオブジェクトに対する操作は、それぞれの操作を記述する API を呼び出すことにより処理されている。

反復解法において、疎行列ベクトル積は計算時間の大半を占めることが多く、性能の最適化が最も重要な処理である。Lis では、多様な行列格納形式について疎行列ベクトル積を実装しており、形式間の変換が可能となっている。以下では、格納形式とそれによって生じる性能の違いについて考察する。

3. 行列格納形式

本節では、Lis で使用できる行列の格納形式について述べる²⁾。行列の行 (列) 番号は 0 から始まるものとする。 $n \times n$ 行列 $A = (a_{ij})$ の非零要素数を nnz とする。

3.1 Compressed Row Storage (CRS)

CRS 形式は行列データを 3 つの配列 (`ptr`, `index`, `value`) に格納する。

- 長さ nnz の倍精度配列 `value` は行列 A の非零要素の値を行方向に沿って格納する。
- 長さ nnz の整数配列 `index` は配列 `value` に格納された非零要素の列番号を格納する。
- 長さ $n + 1$ の整数配列 `ptr` は配列 `value` と `index` の各行の開始位置を格納する。

CRS 形式での格納方法を図 1 に示す。

図 1 の行列 A を 2 プロセス上に CRS 形式で格納すると図 2 のようになる。

3.2 Compressed Column Storage (CCS)

CCS 形式は行列データを 3 つの配列 (`ptr`, `index`, `value`) に格納する。

^{†1} 九州大学情報基盤研究開発センター

Research Institute for Information Technology, Kyushu University

^{*1} <http://www.ssisc.org/lis/>

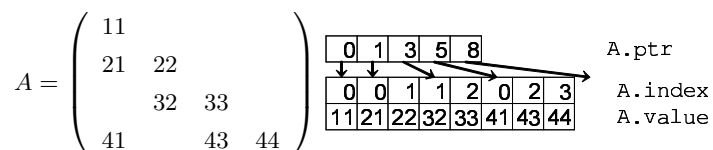


図 1 Data structure of CRS.

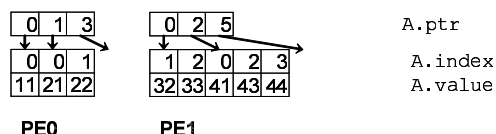


図 2 Data structure of CRS on two processes.

- 長さ nnz の倍精度配列 **value** は行列 A の非零要素の値を列方向に沿って格納する.
- 長さ nnz の整数配列 **index** は配列 **value** に格納された非零要素の行番号を格納する.
- 長さ $n + 1$ の整数配列 **ptr** は配列 **value** と **index** の各列の開始位置を格納する.

CCS 形式での格納方法を図 3 に示す.

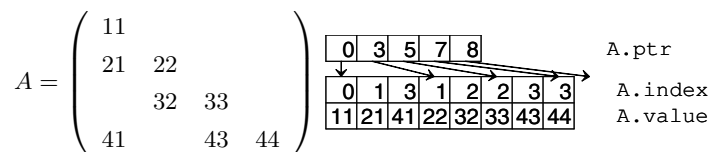


図 3 Data structure of CCS.

2 プロセス上への CCS 形式での格納方法を図 4 に示す.

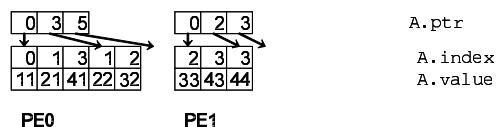


図 4 Data structure of CCS on two processes.

3.3 Modified Compressed Sparse Row (MSR)

MSR 形式は CRS 形式を修正したものである. その違いは対角部分を分けて格納しているところである. MSR 形式は行列データを 2 つの配列 (**index, value**) に格納する. ndz を対角部分の零要素数とする.

- 長さ $nnz + ndz + 1$ の倍精度配列 **value** は n 番目までは行列 A の対角部分を格納する. $n + 1$ 番目の要素は使用しない. $n + 2$ 番目からは行列 A の対角以外の非零要素の値を行方向に沿って格納する.
- 長さ $nnz + ndz + 1$ の整数配列 **index** は $n + 1$ 番目までは行列 A の非対角部分の各行の開始位置を格納する. $n + 2$ 番目からは行列 A の非対角部分の配列 **value** に格納された非零要素の列番号を格納する.

MSR 形式での格納方法を図 5 に示す.

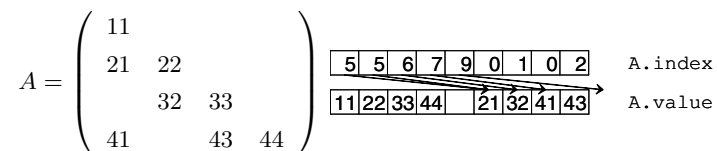


図 5 Data structure of MSR.

2 プロセス上への MSR 形式での格納方法を図 6 に示す.

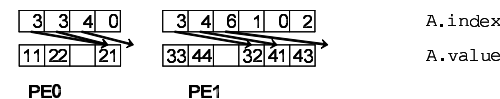


図 6 Data structure of MSR on two processes.

3.4 Diagonal (DIA)

DIA は行列データを 2 つの配列 (**index, value**) に格納する. nnd を行列 A の非零対角の本数とする.

- 長さ $nnd \times n$ の倍精度配列 **value** は行列 A の非零対角を格納する.
- 長さ nnd の整数配列 **index** は主対角から各対角へのオフセットを格納する.

OpenMP 版では以下のように修正している.

DIA は 2 つの配列 (**index**, **value**) に格納する. $nprocs$ をスレッド数とする. nnd_p を行列 A を行ブロック分割した部分行列の非零対角の本数とする. $maxnnd$ を nnd_p の値の最大値とする.

- 長さ $maxnnd \times n$ の倍精度配列 **value** は行列 A を行ブロック分割した部分行列の非零対角を格納する.
- 長さ $nprocs \times maxnnd$ の整数配列 **index** は主対角から各対角へのオフセットを格納する.

DIA 形式での格納方法を図 7 に示す.

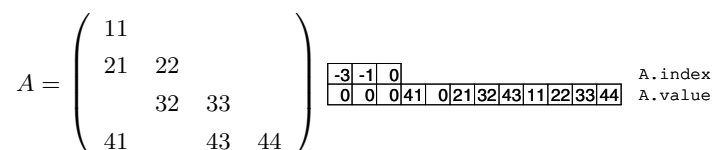


図 7 Data structure of DIA.

2 スレッド上への DIA 形式での格納方法を図 8 に示す.

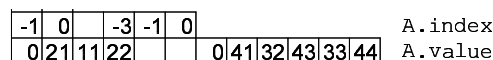


図 8 Data structure of DIA on two threads.

2 プロセス上への DIA 形式での格納方法を図 9 に示す.

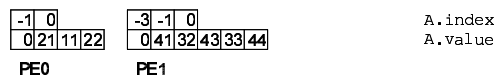


図 9 Data structure of DIA on two processes.

3.5 Ellpack-Itpack generalized diagonal (ELL)

ELL は行列データを 2 つの配列 (**index**, **value**) に格納する. $maxnznr$ を行列 A の各行での非零要素数の最大値とする.

- 長さ $maxnznr \times n$ の倍精度配列 **value** は行列 A の各行の非零要素を列方向に沿って格

納する. 最初の列は各行の最初の非零要素数からなる. ただし, 格納する非零要素数がない場合は 0 を格納する.

- 長さ $maxnznr \times n$ の整数配列 **index** は配列 **value** に格納された非零要素の列番号を格納する. ただし, 第 i 行目の非零要素数を nnz_i とすると $index[nnz_i * n + i]$ にはその行番号 i を格納する.

ELL 形式での格納方法を図 10 に示す.

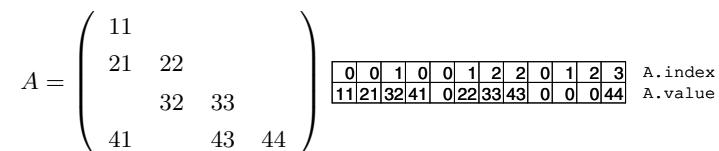


図 10 Data structure of ELL.

2 プロセス上への ELL 形式での格納方法を図 11 に示す.

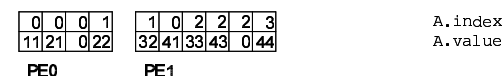


図 11 Data structure of ELL on two processes.

3.6 Jagged Diagonal (JDS)

JDS は最初に各行の非零要素数の大きい順に行の並び替えを行い, 各行の非零要素を列方向に沿って格納する. JDS は行列データを 4 つの配列 (**perm**, **ptr**, **index**, **value**) に格納する. $maxnznr$ を行列 A の各行での非零要素数の最大値とする.

- 長さ n の整数配列 **perm** は並び替えた行番号を格納する.
- 長さ nnz の倍精度配列 **value** は並び替えられた行列 A のぎざぎざ対角を格納する. 最初のぎざぎざ対角は各行の最初の非零要素数からなる. 次のぎざぎざ対角は各行の 2 番目の非零要素からなる. これを順次繰り返していく.
- 長さ nnz の整数配列 **index** は配列 **value** に格納された非零要素の列番号を格納する.
- 長さ $maxnznr + 1$ の整数配列 **ptr** は各ぎざぎざ対角の開始位置を格納する.

OpenMP 版では以下のように修正を行っている.

JDS は 4 つの配列 (**perm**, **ptr**, **index**, **value**) に格納する. $nprocs$ をスレッド数とする.

$maxn_zr_p$ を行列 A を行ブロック分割した部分行列の各行での非零要素数の最大値とする。
 $maxmaxn_zr$ は配列 $maxn_zr_p$ の値の最大値である。

- 長さ n の整数配列 **perm** は行列 A を行ブロック分割した部分行列を並び替えた行番号を格納する。
- 長さ nnz の倍精度配列 **value** は並び替えられた行列 A のぎざぎざ対角を格納する。最初のぎざぎざ対角は各行の最初の非零要素数からなる。次のぎざぎざ対角は各行の2番目の非零要素からなる。これを順次繰り返していく。
- 長さ nnz の整数配列 **index** は配列 **value** に格納された非零要素の列番号を格納する。
- 長さ $nprocs \times (maxmaxn_zr + 1)$ の整数配列 **ptr** は行列 A を行ブロック分割した部分行列の各ぎざぎざ対角の開始位置を格納する。

JDS 形式での格納方法を図 12 に示す。

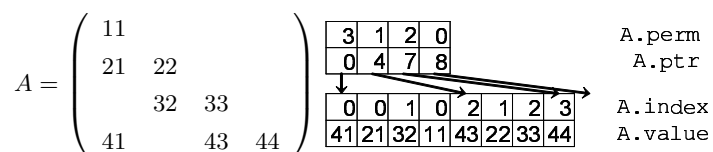


図 12 Data structure of JDS.

2 スレッド上への JDS 形式での格納方法を図 13 に示す。

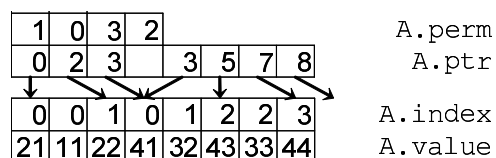


図 13 Data structure of JDS on two threads.

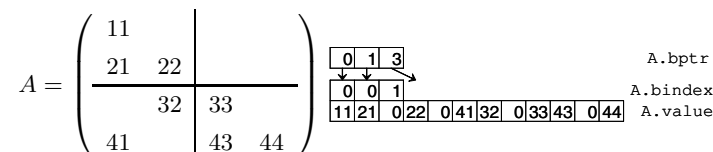
2 プロセス上への JDS 形式での格納方法を図 14 に示す。

3.7 Block Sparse Row (BSR)

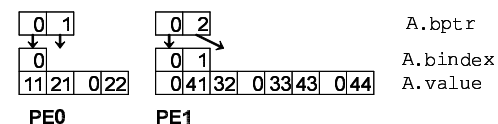
BSR では行列を $r \times c$ の大きさの部分行列 (ブロックと呼ぶ) に分解する。BSR は CRS と同様の手順で非零ブロック (少なくとも1つの非零要素が存在する) を格納する。 $nr = n/r$, $nnzb$ を A の非零ブロック数とする。BSR は行列データを3つの配列 (**bp**tr, **bin**dex, **value**) に格納する。

value) に格納する。

- 長さ $nnzb \times r \times c$ の倍精度配列 **value** は非零ブロックの全要素を格納する。
 - 長さ $nnzb$ の整数配列 **bin**dex は非零ブロックのブロック列番号を格納する。
 - 長さ $nr + 1$ の整数配列 **bp**tr は配列 **bin**dex のブロック行の開始位置を格納する。
- BSR 形式での格納方法を図 15 に示す。



2 プロセス上への BSR 形式での格納方法を図 16 に示す。



3.8 Block Sparse Column (BSC)

BSC では行列を $r \times c$ の大きさの部分行列 (ブロックと呼ぶ) に分解する。BSC は CCS と同様の手順で非零ブロック (少なくとも1つの非零要素が存在する) を格納する。 $nc = n/c$, $nnzb$ を A の非零ブロック数とする。BSC は行列データを3つの配列 (**bp**tr, **bin**dex, **value**) に格納する。

- 長さ $nnzb \times r \times c$ の倍精度配列 **value** は非零ブロックの全要素を格納する。

- 長さ $nnzb$ の整数配列 **bindex** は非零ブロックのブロック行番号を格納する.
 - 長さ $nc + 1$ の整数配列 **bptr** は配列 **bindex** のブロック列の開始位置を格納する.
- BSC 形式での格納方法を図 17 に示す.

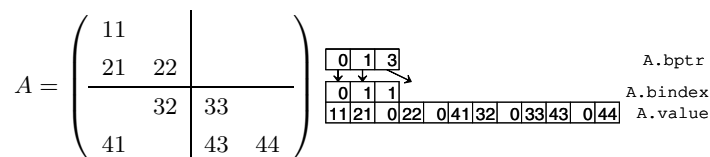


図 17 Data structure of BSC.

2 プロセス上への BSC 形式での格納方法を図 18 に示す.

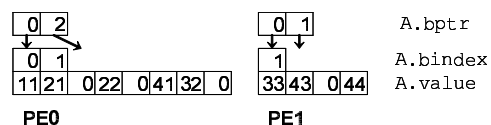


図 18 Data structure of BSC on two processes.

3.9 Variable Block Row (VBR)

VBR 形式は BSR 形式を一般化したものである. 行と列の分割位置は配列 (**row**, **col**) で与えられる. VBR は CRS と同様の手順で非零ブロック (少なくとも 1 つの非零要素が存在する) を格納する. nr , nc をそれぞれ行分割数, 列分割数とする. $nnzb$ を A の非零ブロック数, nnz を非零ブロックの全要素数とする. VBR は行列データを 6 つの配列 (**bptr**, **bindex**, **row**, **col**, **ptr**, **value**) に格納する.

- 長さ $nr + 1$ の整数配列 **row** はブロック行の開始行番号を格納する.
 - 長さ $nc + 1$ の整数配列 **col** はブロック列の開始列番号を格納する.
 - 長さ $nnzb$ の整数配列 **bindex** は非零ブロックのブロック列番号を格納する.
 - 長さ $nr + 1$ の整数配列 **bptr** は配列 **bindex** のブロック行の開始位置を格納する.
 - 長さ nnz の倍精度配列 **value** は非零ブロックの全要素を格納する.
 - 長さ $nnzb + 1$ の整数配列 **ptr** は配列 **value** の非零ブロックの開始位置を格納する.
- VBR 形式での格納方法を図 19 に示す.

2 プロセス上への VBR 形式での格納方法を図 20 に示す.

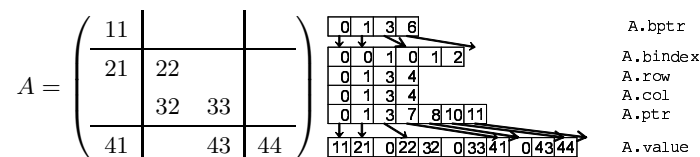


図 19 Data structure of VBR.

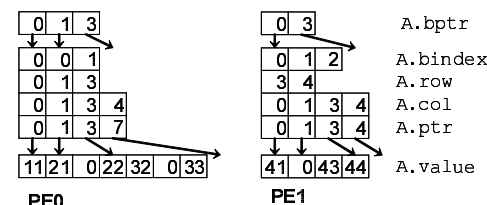


図 20 Data structure of VBR on two processes.

3.10 Coordinate (COO)

COO は行列データを 3 つの配列 (**row**, **col**, **value**) に格納する.

- 長さ nnz の倍精度配列 **value** は非零要素を格納する.
- 長さ nnz の整数配列 **row** は非零要素の行番号を格納する.
- 長さ nnz の整数配列 **col** は非零要素の列番号を格納する.

COO 形式での格納方法を図 21 に示す.

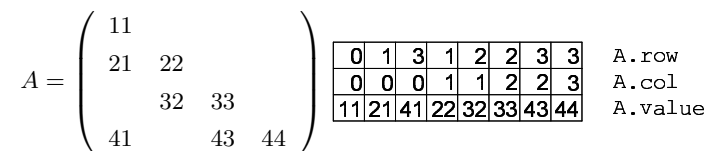


図 21 Data structure of COO.

2 プロセス上への COO 形式での格納方法を図 22 に示す.

3.11 Dense (DNS)

DNS は行列データを 1 つの配列 (**value**) に格納する.

- 長さ $n \times n$ の倍精度配列 **value** は列優先で要素を格納する.

表 2 Performance of spmvtest2 on Xeon 5570 server.

Problem Size	40,000	80,000	160,000	320,000
# MPI processes	1	2	4	8
CRS (MFLOPS)	1110	2190	4140	<u>4440</u>
DIA (MFLOPS)	1630	3190	<u>5010</u>	4370
ELL (MFLOPS)	1210	2400	<u>4270</u>	3770
JDS (MFLOPS)	1030	2030	<u>3370</u>	3040
Problem Size	40,000	80,000	160,000	320,000
# threads	1	2	4	8
CRS (MFLOPS)	1190	2380	2160	<u>3780</u>
DIA (MFLOPS)	1640	3130	<u>6040</u>	4940
ELL (MFLOPS)	1250	2440	<u>4350</u>	3560
JDS (MFLOPS)	1020	2030	<u>3070</u>	2860

表 4 Performance of spmvtest2 on Hitachi SR16000.

Problem Size	40,000	80,000	160,000	320,000	640,000	1,280,000
# MPI processes	1	2	4	8	16	32
CRS (MFLOPS)	580	851	1680	3270	6430	<u>13400</u>
DIA (MFLOPS)	2130	2370	4000	7730	14900	<u>30600</u>
ELL (MFLOPS)	1790	1750	2440	4860	9490	<u>19100</u>
JDS (MFLOPS)	1390	1330	2020	4020	7960	<u>16200</u>
# threads	1	2	4	8	16	32
CRS (MFLOPS)	580	1090	2210	4390	8740	<u>15100</u>
DIA (MFLOPS)	2130	3930	7500	15000	28700	<u>44100</u>
ELL (MFLOPS)	1790	2930	5900	11500	23000	<u>33800</u>
JDS (MFLOPS)	1390	2250	4550	9060	17600	<u>27300</u>

表 3 Performance of spmvtest1 on Hitachi SR16000.

Problem Size	40,000	80,000	160,000	320,000	640,000	1,280,000
# MPI processes	1	2	4	8	16	32
CRS (MFLOPS)	310	484	976	1960	3730	<u>7760</u>
DIA (MFLOPS)	1800	1940	3900	7800	15600	<u>31200</u>
ELL (MFLOPS)	1530	1490	2130	6070	12100	<u>24000</u>
JDS (MFLOPS)	1150	1090	1480	4270	8450	<u>17100</u>
# threads	1	2	4	8	16	32
CRS (MFLOPS)	310	607	1210	2420	4790	<u>9380</u>
DIA (MFLOPS)	1800	3270	6470	12600	25900	<u>43000</u>
ELL (MFLOPS)	1530	2880	5620	11100	21000	<u>40700</u>
JDS (MFLOPS)	1150	2090	4080	7810	15100	<u>29300</u>

表 5 Performance of spmvtest1 on NEC SX-9.

Problem Size	40,000	80,000	160,000	320,000	640,000
# MPI processes	1	2	4	8	16
CRS (MFLOPS)	5.39	10.8	21.6	<u>43.3</u>	-
DIA (MFLOPS)	9780	11600	20800	<u>42600</u>	-
ELL (MFLOPS)	1320	2380	4730	<u>9270</u>	-
JDS (MFLOPS)	835	1510	3010	<u>5860</u>	-
# threads	1	2	4	8	16
CRS (MFLOPS)	5.44	11.0	14.4	43.7	<u>87.4</u>
DIA (MFLOPS)	8930	9460	18000	35600	<u>64700</u>
ELL (MFLOPS)	1280	2300	4470	9150	<u>18000</u>
JDS (MFLOPS)	828	1550	1040	6160	<u>12100</u>

表 6 Performance of spmvtest2 on NEC SX-9.

Problem Size	40,000	80,000	160,000	320,000	640,000
# MPI processes	1	2	4	8	16
CRS (MFLOPS)	13.2	26.5	52.7	<u>106</u>	-
DIA (MFLOPS)	10400	12100	16000	<u>29600</u>	-
ELL (MFLOPS)	1310	2410	4450	<u>8720</u>	-
JDS (MFLOPS)	1070	1890	3460	<u>6340</u>	-
# threads	1	2	4	8	16
CRS (MFLOPS)	13.4	27.0	53.6	-	<u>212</u>
DIA (MFLOPS)	11700	15500	28300	-	<u>100000</u>
ELL (MFLOPS)	1310	2530	4900	-	<u>19500</u>
JDS (MFLOPS)	1080	2050	4090	-	<u>16000</u>

5. 考 察

本稿では、現在開発を進めている並列反復解法ライブラリ Lis への適用を目的として、複数アーキテクチャ上での疎行列ベクトル積の事前性能評価の妥当性について検討した。疎行列ベクトル積の性能は、行列の形状、並列化手法、メモリの階層構造等によって大きく性能が変化するため、すべての場合に最適な解法を見出すのは難しい。したがって、現実的な解として、本研究で試みた局所的な性能解析を一般化し、より多様なデータ格納手法について評価を可能にすることが考えられる。また、解析手法の自動化により、性能可搬性の確保を容易にすることも重要である。本研究では、今後これらの手法について、実装、評価を行っていく予定である。

参 考 文 献

- 1) Dongarra, J.: Freely Available Software for the Solution of Linear Algebra Problems, <http://www.netlib.org/utk/people/JackDongarra/la-sw.html> (2009).
- 2) The Scalable Software Infrastructure Project: Lis 1.2.15 User's Manual, <http://www.ssisc.org/lis/> (2009).