RIKEN
Center for
Computational Science

2025 年 7 月 30 日

有限要素解析のための並列ソルバーに関するスクール 2025

疎行列による Krylov 部分空間法の実装

鈴木 厚 [1]

[1] 理研計算科学研究センター 大規模並列数値計算技術研究チーム
atsushi.suzuki.aj@a.riken.jp

疎行列格納方法：1/3

n : # 行

nnz : # 非零要素

$[A]_{i\,j}$: $(i,\ j)$ 成分にある非零要素の値

- COO (Coordinate) 形式 `MUMPS`

```
struct COOformat {
  int n, nnz;
  int irow[nnz];
  int jcol[nnz];
  double coef[nnz];
};
```

- CSR (Compressed Sparse Row) /
  CRS (Compressed Row Storage) 形式 `Pardiso`

```
struct CSRformat {
  int n, nnz;
  int ptrow[n+1];
  int indcol[nnz];
  double coef[nnz];
}
```

$[A]_{i\,j}$=coef[k]
j=indcol[k], ptrow[i] $\leq k <$ ptrow[$i+1$]

疎行列格納方法：2/3

5×5 の非対称行列の例：$n = 5, nnz = 15$.

1.1	1.2		1.4	
2.1	2.2	2.3		2.5
	3.2	3.3		
4.1			0.0	4.5
	5.2		5.4	5.5

	0	1	2	3	4
0	0	1		2	
1	3	4	5		6
2		7	8		
3	9			10	11
4		12		13	14

i	0			1				2		3			4			5
ptrow[i]	0			3				7		9			12			15
indcol[k]	0	1	3	0	1	2	4	1	2	0	3	4	1	3	4	
coef[k]	1.1	1.2	1.4	2.1	2.2	2.3	2.5	3.2	3.3	4.1	0.0	4.5	5.2	5.4	5.5	

- 値が 0 であっても対角成分を記憶する
- `indcol[]` はそれぞれの行毎に昇順

疎行列格納方法：3/3

5×5 の対称行列の上三角部分の例：$n = 5, nnz = 10$.

						0	1	2	3	4
1.1	1.2		1.4		0	0	1		2	
	2.2	2.3		2.5	1		3	4		5
		3.3			2			6		
			0.0	4.5	3				7	8
				5.5	4					9

i	0			1			2	3		4	5
ptrow[i]	0			3			6	7		9	10
indcol[k]	0	1	3	1	2	4	2	3	4	4	
coef[k]	1.1	1.2	1.4	2.2	2.3	2.5	3.3	0.0	4.5	5.5	

- 値が 0 であっても対角成分を記憶する
- `indcol[]` はそれぞれの行毎に昇順
- 上三角行列の保存形式は `Pardiso` 直接法で採用されている

SpMV：疎行列ベクトル積演算：1/3

A : CSRformat { ptrow, indcol, coef }, $\vec{x}, \vec{y} \in \mathbb{R}^N$
$\vec{y} = A\,\vec{x}$ の計算は次の様に実行される

$$
\begin{aligned}
[A\vec{x}]_i &= \sum_j [A]_{i\,j}[\vec{x}]_j \\
&= \sum_{j \in \{j\,;\,[A]_{i\,j} \neq 0\}} [A]_{i\,j}[\vec{x}]_j \\
&= \sum_{\texttt{ptrow}[i] \leq k < \texttt{ptrow}[i+1]} \texttt{coef}[k] \times [\vec{x}]_{\texttt{indcol}[k]}
\end{aligned}
$$

```
for (i = 0; i < n; i++) {
  y[i] = 0.0;
  for (k = ptrow[i]; k < ptrow[i + 1]; k++) {
    int j = indcol[k];
    y[i] += coef[k] * x[j];
  }
}
```

SpMV：疎行列ベクトル積演算：2/3

A：CSRformat { ptrow, indcol, coef }, 対称行列のうち上三角部分を記憶,
$\vec{x}, \vec{y} \in \mathbb{R}^N$
$\vec{y} = A\vec{x}$ の演算を実行するため次を仮定する

- 行列 A の対角成分を記憶する
- i-行目の最初の非零成分は対角である：`indcol[ptrow[i]] == i`

```
for (i = 0; i < n; i++) {
  y[i] = 0.0;
}
for (i = 0; i < n; i++) {
  for (k = ptrow[i] + 1; k < ptrow[i + 1]; k++) {
    int j = indcol[k];
    y[i] += coef[k] * x[j];
    y[j] += coef[k] * x[i];
  }
  int k = ptrow[i];
//    int j = indcol[k] ( == i )
  y[i] += coef[k] * x[i];
}
```

SpMV：疎行列ベクトル積演算：3/3

スカラ α と β に対して CSRformat で格納している一般の非対称行列 A をベクトルに作用させる演算 $y = \alpha A\vec{x} + \beta\vec{y}$

```
void SparseGEMV(struct CSRformat &A,
                const double &alpha, std::vector<double> &x,
                const double &beta, std::vector<double> &y)
{
  int nrow = A.n;
  for (int i = 0; i < nrow; i++) {
    y[i] *= beta;
  }
  double tmp;
  for (int i = 0; i < nrow; i++) {
    tmp = 0.0;
    for (k = ptrow[i]; k < ptrow[i + 1]; k++) {
      int j = indcol[k];
      tmp += coef[k] * x[j];
    }
  }
  y[i] += alpha * tmp;
}
```

内積演算

二つのベクトル：$\vec{x},\ \vec{y} \in \mathbb{R}^N$ に対して

$$(\vec{x}, \vec{y}) = \sum_{1 \le i \le N} [\vec{x}]_i [\vec{y}]_i$$

```
double tmp = 0.0;
for (i = 0; i < n; i++) {
  tmp += x[i] * y[i];
}
```

BLAS レベル 1 サブルーチン ddot

```
double cblas_ddot(const int n,
                  const double *x, const int incx,
                  const double *y, const int incy);

tmp = cblas_ddot(n, &x[0], 1, &y[0], 1);
```

C++ STL std::vector< > クラス

```
#include <vector>
std::vector<double> x(100);  // allocation with size = 100
x.resize(200); // enlarging array with keeping 100 entries
x.clean();     // deallocation
&x[0];         // double pointer to the first entry
x.data();      //
```

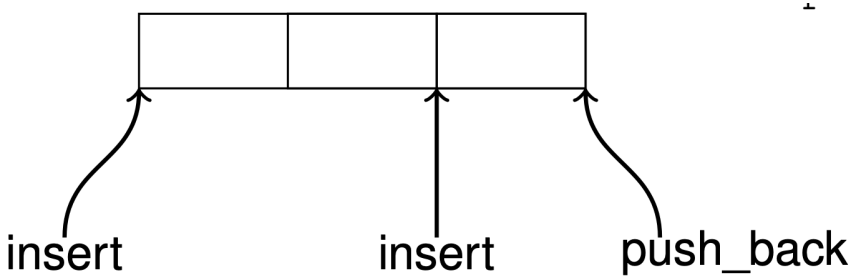
疎行列の格納形式の変換：COO 形式から CSR 形式へ：1/2

5×5 の非対称行列の例：$n = 5, nnz = 15$.

1.1	1.2		1.4	
2.1	2.2	2.3		2.5
	3.2	3.3		
4.1			0.0	4.5
	5.2		5.4	5.5

	pcol				pcoef			
0	0	1	3		1.1	1.2	1.4	
1	0	1	2	4	2.1	2.2	2.3	2.5
2	1	2			3.2	3.3		
3	0	3	4		4.1	0.0	4.5	
4	1	3	4		5.2	5.4	5.5	

列インデックスは i-行毎に `list<int> pcol[i]` に格納



- それぞれの行の配列は `vector<list<int> >` に格納
- 昇順になるようにデータは `std::list` のイテレータを用いる
- 一次元配列データ `int indcol[k]` は `pcol` より複写して作製

リスト構造は動的データの扱いに適するが HPC アプリケーションでは効率的でない

疎行列の格納形式の変換：COO 形式から CSR 形式へ：2/2

- 各行の記憶のため動的なメモリー確保が不可欠

```
std::vector<std::list<int> > pcol(nrow);
std::vector<std::list<double> > pcoef(nrow);
for (int k = 0; k < nnz; k++) {
  int i = Acoo.irow[k], j = Acoo.jcol[k];
  double coef = Acco.coef[k];
  if (pcol[i].empty())
    pcol[i].push_back(j); pcoef[i].push_back(coef);
  else {
    if (pcol[i].back() < j)
      pcol[i].push_back(j); pcoef[i].push_back(coef);
    else {
      std::list<double>::iterator iv = pcoef[i].begin();
      std::list<int>::iterator it = pcol[i].begin();
      for (; it != pcol[i].end(); ++it, ++iv) {
        if ((*it) > j)
          pcol[i].insert(it, j); pcoef[i].insert(iv, coef);
      }  } }
}
Acsr.ptrow[0] = 0;
int k = 0;
for (int i = 0; i < n; i++) {
  ptrow[i + 1] = ptrow[i] + pcol[i].size();
  std::list<double>::iterator iv = pcoef[i].begin();
  std::list<int>::iterator it = pcol[i].begin();
  for (; it != pcol[i].end(); ++it, ++iv, k++) {
    Acsr.pcol[k] = (*it); Acsr.coef[k] = (*iv); } }
```

疎行列の格納形式の変換：CSR 形式から COO 形式へ

```
structure COOformat {
  int n, nnz;
  int *irow;
  int *jcol;
  double *coef;
};
structure CSRformat {
  int n, nnz;
  int *ptrow;
  int *indcol;
  double *coef;
}
int n, nnz;      //   peparatoin of Acsr.ptrow, Acsr.indcol
CSRformat Acsr;
Acsr.n = n;  Acsr.nnz = nnz;
Acsr.ptrow = new int[n + 1];  Acsr.indcol = new int[nnz];

COOfromat Acoo;
Acoo.irow = new int[nnz];       Acoo.jcol = new int[nnz];
//
Acoo.n = Acsr.n; Acoo.nnz = nnz;
for (int i = 0; i < n; ++i) {
  for (int k = Acsr.ptrow[i]; k < Acsr.ptrow[i + 1]; ++k) {
    Acoo.irow[k] = i;
    Acoo.jcol[k] = Acsr.indcol[k];
    Acoo.coef[k] = Acsr.coef[k];
  }
}
```

PETSc ライブラリー：1/7

PETSc : the Portable, Extensible Ttoolkit for Scientifc computation
合衆国 Argonne 研究所で開発された偏微分方程式の線形/非線形方程式の高い並列性を持つ解法 (scalable parallel solution) のためのデータ構造と演算ルーチンの集合体 cf. Ed. Bueler, PETSc for Partial Differential Equations, 2020, SIAM
doi.org/10.1137/1.9781611976311

- `Vec`：ベクトルオブジェクト (分散データ)
- `Mat`：行列オブジェクト, 密/疎行列 (分散データ)
- `KSP`：Krylov 部分空間法ソルバー
- `PC`：前処理
- `PetscInt`：整数 32 ビット (デフォルト) / 64 ビット
- `PetscReal`：実数の浮動小数点
- `PetscComplex`：複素数の浮動小数点

ベクトルオブジェクトは MPI による分散メモリーで確保される

```
Vec x;
PetscInt i[4] = {0, 1, 2, 3};
PetscReal v[4] = {1.0, -1.0, 2.0, -4.0};
VecCreate(PETSC_COMM_WORLD, &x);
VecSetSizes(x, PETSC_DECIDE, 10000); // (x, 500, 10000) for 20 procs
VecSetFromOptions(x);
VecSetValues(x, 4, i, v, INSETERT_VALUES);  // put first 4 entries
VecAssemblyBegin(x);
VecAssemblyEnd(x);
VecView viewer;
PetscViewerASCIIOpen(PETSC_COMM_WORLD, "vec.data", viewer);
VecView(x, viewer);
PetscViewerDestory(viewer);
```

PETSc ライブラリー：2/7

ksp/tutorial/ex1.c 行列の設定と KSP ソルバーの例

```
#include <petscksp.h>

int main(int argc, char **args)
{
  Vec         x, b, u; /* approx solution, RHS, exact solution */
  Mat         A;       /* linear system matrix */
  KSP         ksp;     /* linear solver context */
  PC          pc;      /* preconditioner context */
  PetscInt    i, n = 10, col[3], its;
  PetscMPIInt size;
  PetscScalar value[3];

  PetscInitialize(&argc, &args, (char *)0, help);
  MPI_Comm_size(PETSC_COMM_WORLD, &size);

  PetscOptionsGetInt(NULL, NULL, "-n", &n, NULL);
  // set up for vectors
  MatCreate(PETSC_COMM_SELF, &A);
  MatSetSizes(A, PETSC_DECIDE, PETSC_DECIDE, n, n);
  MatSetFromOptions(A);
  MatSetUp(A);
```

PETSc ライブラリー：3/7

ksp/tutorial/ex1.c 行列の設定と KSP ソルバーの例

```
value[0] = -1.0;
value[1] = 2.0;
value[2] = -1.0;
for (i = 1; i < n - 1; i++) {
  col[0] = i - 1;
  col[1] = i;
  col[2] = i + 1;
  MatSetValues(A, 1, &i, 3, col, value, INSERT_VALUES);
}
// two column data for i = 0 and i =n
//  MatSetValues(A, 1, &i, 2, col, value, INSERT_VALUES);
//
MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY);

VecSet(u, 1.0);
MatMult(A, u, b);

KSPCreate(PETSC_COMM_SELF, &ksp);
KSPSetOperators(ksp, A, A);
KSPGetPC(ksp, &pc);
PCSetType(pc, PCJACOBI);
KSPSetTolerances(ksp, 1.e-5, PETSC_DEFAULT,
                 PETSC_DEFAULT, PETSC_DEFAULT);
KSPSetFromOptions(ksp);
KSPSolve(ksp, b, x);
```

PETSc ライブラリー：4/7

- `MatSetVaules()`関数の後に`MatAssenblyBegin()` と `MatAssenblyEnd()` を呼ぶ必要がある
- `PCSetType` 関数は前処理の種類 `PCType` を設定する
- `PCType` は幾つかの前処を提供
  `PCJACOBI`, `PCILU`, `PCSOR`, `PCGAMG`, `PCASM`

KSPSetFromOptions() はコマンドラインから Krylov 部分空間法のオプションを指定する.

- 反復回数の指定や収束の途中経過の表示
  `-ksp_max_it 100`
  `-ksp_rtol 1.0e-6`
  `-ksp_monitor`
  `-ksp_view`
  `-ksp_converged_reason`
- Krylov 部分空間法ソルバーの選択 `-ksp_type`
  `cg`, `gmres`, `gcr`, `minres`, `bcgs`,
- 前処理の選択 `-pc_type`
  `jacobi`, `ilu`, `sor`, `gamg`, `asm`, `hypre`
- パフォーマンスサマリーの表示
  `-log_view`

PETSc ライブラリー：5/7

CSR データを一括して PETSc に渡す.

- CSR 形式のデータにあらかじめ変換する

```
struct csr_matrix {
  PetscInt nrow, nnz;
  std::vector<PetscInt> ia, ja;
  std::vector<PetscReal> coefs;
};
```

- MatCreateSeqAIJWithArrays を用いて csr_matrix A → Mat A

```
MatCreateSeqAIJWithArrays(PETSC_COMM_SELF,
                          nrow, nrow,
                          &a.ia[0], &a.ja[0], &a.coefs[0], &A);
MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY);
KSPCreate(PETSC_COMM_WORLD, &ksp);
KSPSetOperators(ksp, A, A);
KSPGetPC(ksp, &pc);
PCSetType(pc, PCJACOBI); // diagonal scaling
KSPSetFromOptions(ksp);
KSPSolve(ksp, b, x);
```