RIKEN Center for Computational Science

2024 年 6 月 27 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 2024

加法的 Schwarz 前処理の実装

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

METIS を用いた重なりの無い部分行列への分解

```
int nrow, nnz;
int xadj[nrow];          // connectivity of the sparse matrix
int adjcy[nnz - nrow]; // excluding diagonal from CSR
int part[nrow];
int ncon = 1, objval;
idx_t options[METIS_NOPTIONS] = {0} ;
METIS_SetDefaultOptions(options);
options[METIS_OPTION_NUMBERING] = 0;
options[METIS_OPTION_DBGLVL] = METIS_DBG_INFO;
METIS_PartGraphRecursive(&nrow, &ncon, xadj, adjcy,
                         NULL, NULL, NULL,
                         &nparts,
                         NULL, NULL,
                         options, &objval, part);
```

添字集合 $\Lambda = \{1, 2, \cdots, N\}$ は重なりのない分割の添字集合 Λ_p の和集合からなる

$$\Lambda = \oplus_{1 \le p \le P} \Lambda_p \quad \Lambda_p \cap \Lambda_q = \emptyset \quad (p \neq q)$$

配列 part[k] $1 \le k \le N$ 添字 $1 \le p \le P$ の部分行列への対応を表わす

$$\Lambda_p = \{k \in \{1, \cdots, N\};\ \texttt{part}[k] = p\}$$

重なりの無い分割から重なりのある分割の生成

重なりの無い分割 $\Lambda = \oplus_{1 \le p \le P} \Lambda_p \quad \Lambda_p \cap \Lambda_q = \emptyset$ の部分行列の添字集合
$\Lambda_p^{(0)} = \Lambda_p$ から始めて l 回目から $l+1$ 回目の集合

$$\Lambda_p^{(l+1)} = \{ j \, ; \, [A]_{i\,j} \neq 0 \;\; i \in \Lambda_p^{(l)} \}$$

を生成する：$\Lambda_p = \Lambda_p^{(0)} \subset \Lambda_q^{(1)} \subset \cdots$
この操作はマスクベクトルを更新することで実現する

```
structure CSRformat acsr;
std::vector<std::vector<int> > mask[nparts];
for (int n = 0; n < nparts; n++) mask[n].resize(nrow, 0);

for (int i = 0; i < nrow; i++)
  mask[part[i]][i] = 1;          // masking domain n == part[i]

for (int n = 0; n < nparts; n++) {
  for (int ll = 0; ll < noverlap; ll++) {
    std::vector<int> itmp(mask[n]);   // copy mask[n] to itmp
    for (int i = 0; i < nrow; i++) {
      if (itmp[i] == 1) {
        for (int k = acsr.ptrow[i]; k < acsr.ptrow[i + 1]; k++)
          mask[n][acsr.indcol[k]] = 1;
      }
    }
  }
}
```

単位の分解の構成

離散的な単位の分解

$$\sum_{p=1}^{P} R_p^T D_p R_p = I_N,$$

$$[D_p]_{kk} = \begin{cases} 1 & k \in \Lambda_p,\ k \notin \Lambda_q, \forall q \neq p, \\ 1/\#\{p; k \in \Lambda_p\} & \text{それ以外} \end{cases}$$

重なりのある添字分割を保持する mask 配列から, 制限作用素 R_p と重み D_p を次のように生成する

```
std::vector<std::vector<int> > local2global(nparts);
std::vector<double> weightPTU(nrow, 0.0);

for (int n = 0; n < nparts; n++) {
  for (int i = 0; i < nrow; i++) {
    if (mask[n][i] == 1) {
      local2globall[n].push_back(i);
      weightPTU[i] += 1.0;
    }
  }
}
for (int i = 0; i < nrow; i++) {
  weightPTU[i] = 1.0 / weightPTU[i];
}
```

全体行列から添字 n の部分行列を抜きだすことは mask[n] ベクトルと写像 local2global[n] によって行う.

重なりのある部分領域への疎行列のデータ分散：1/5

添字 p の行列は制限作用素 R_p から次のように生成される

$$A_p = R_p A R_p^T, \quad A = \sum_p R_p^T D_p A_p R_p$$

行列-ベクトル積は部分行列 A_p 毎の積の重み付きの和から計算される

$$\begin{aligned} \vec{y} = A\vec{x} &= \sum_p R_p^T D_p A_p R_p \vec{x} \\ &= \sum_p R_p^T D_p A_p \vec{x}_p \end{aligned}$$

$\vec{x}_p$：は添字集合 Λ_p の部分ベクトル

- 部分行列での疎行列-ベクトル積演算 (SpMV) $\vec{y}_p = A_p \vec{x}_p$
- 重み D_p 付き総和計算, $\vec{y} = \sum_p R_p^T D_p \vec{y}_p$

総和計算には異るプロセッサに割り当てられたデータの通信が必要

重なりのある部分領域への疎行列のデータ分散：2/5

重み付き総和計算 $\vec{y} = \sum_p R_p^T D_p \vec{y}_p$ の最も簡単な実装方法

- $\vec{y}_p = A_p \vec{x}_p$ を部分行列毎に実行する
- 長さ N の全体配列 $\vec{z}_p \in \mathbb{R}^N$ を準備して部分添字集合 Λ_p 以外を零で埋める

$$
\begin{aligned}
[\vec{z}_p]_{i \neq \Lambda_p} &= 0 \\
[\vec{z}_p]_{\lambda_p(j)} &= [\vec{y}_p]_j \quad \lambda_p : \{1, \cdots, N_p\} \to \Lambda_p \subset \{1, \cdots, N\}
\end{aligned}
$$

プロセッサー全にあるデータ $\vec{z}_p$ を全て足し込む総和演算を実行する

```
MPI_Allreduce(z_p, y, nrow, MPI_DOUBLE, MPI_SUM,
              MPI_COMM_WORLD);
```

部分行列の内部にある (部分行列の重なりのある部分を除いた) データの添字は $\Lambda_{p,I} \cap \Lambda_q = \emptyset \ (\forall q \neq p)$ と表現されるが, そこでは他の部分行列での値は 0 として和の演算が実行される

重なりのある部分領域への疎行列のデータ分散：3/5

部分行列の内部で 0 を足し込む不要な演算を避けるために
部分行列の添え字集合を部分行列内部と境界に分割する

$$\Lambda_p = \Lambda_{p,I} \oplus \Lambda_{p,B} \quad \Lambda_{p,I} \cap \Lambda_q = \emptyset \quad \forall q \neq p \ \exists r \Lambda_{p,B} \cap \Lambda_r \neq \emptyset$$

この分割により総和計算は部分行列の境界で実行して, 部分行列内部はそのまま
データを移動するだけで済むことが分る

$$\vec{y} = \sum_p R_p^T D_p \vec{y}_p$$

$$= \left(\{ R_{p,I}^T D_{p,I} \vec{y}_{p,I} \}, \sum_q R_{q,B}^T D_{q,B} \vec{y}_{q,B} \right)$$

部分行列の添え字集合の内部と境界への分割 $\Lambda_p = \Lambda_{p,I} \cap \Lambda_{p,B}$ は単位の分解により設定できる

$$i \in \Lambda_{p,I} \quad \Leftrightarrow \quad [D_p]_i = 1$$

$$i \in \Lambda_{p,B} \quad \Leftrightarrow \quad [D_p]_i < 1$$

```
std::vector<std::vector<int> > localI2global(nparts);
std::vector<std::vector<int> > localB2global(nparts);
// n fixed
for (std::vector<int>::iterator it = local2global[n].begin();
  it != local2global[n].end(); ++it) {
  if (weightPTU((*it) == 1.0)
    localI2global[n].push_back((*it));
  else
    localB2global[n].push_back((*it));
}
```

data distribution of sparse matrix with overlapping subdomains : 4/5

近接間通信による部分領域毎のデータの更新

$$\Lambda_p = \Lambda_{p,I} \oplus \Lambda_{p,B} \quad \Lambda_{p,I} \cap \Lambda_q = \emptyset \quad \forall q \neq p \text{ に追加して } \exists r\ \Lambda_{p,B} \cap \Lambda_r \neq \emptyset$$

$$\Lambda_{p,r} = \Lambda_{p,B} \cap \Lambda_{r,B} \quad p \text{ と } r \text{ の間でのデータ転送}$$

部分領域とその境界の添え字の間の対応

$$\{1, \cdots, N_p\} \ni i \mapsto \lambda_p(i) \in \Lambda_p \subset \Lambda \qquad \text{全体}$$

$$\{1, \cdots, N_{p,B}\} \ni i \mapsto \lambda_{p,B}(i) \in \Lambda_{p,B} \subset \Lambda_p \qquad \text{全体}$$

$$\{1, \cdots, N_{p,B}\} \ni i \mapsto \widehat{\lambda}_{p,B}(i) \in \{1, \cdots, N_p\} \quad \widehat{\Lambda}_{p,B} = \{\widehat{\lambda}_{p,B}(i)\,;\, i \in \{1, \cdots, N_{p,B}\}\}$$

$\widehat{\Lambda}_{p,r} \subset \widehat{\Lambda}_{p,B}$: $\widehat{\Lambda}_{r,B}$ のデータを共有する境界での部分添字

- 領域 p でデータ $\widehat{\Lambda}_{p,r}$ を準備
- データを領域 r へ送信
- 領域 r からのデータを受信
- 領域 p でデータ $\widehat{\Lambda}_{p,r}$ を展開

```
MPI_Isend( &senddata[0], nsenddata, MPI_DOUBLE, destrank, 0,
           MPI_COMM_WORLD, &requests[0]);
MPI_Irecv( &recvdata[0], nsenddata, MPI_DOUBLE, srcrank, 0,
           MPI_COMM_WORLD, &requests[1]);
MPI_Waitall( 2, &requests[0], &statuses[0]);
```

重なりのある部分領域への疎行列のデータ分散：5/5

内積演算は単位の分解の重みよりベクトルのデータの移動は不要

$$
\begin{aligned}
(\vec{x}, \vec{y}) =& (\vec{x}, \sum_p R_p^T D_p \vec{y}_p) \\
=& \sum_p (R_p \vec{x}, D_p \vec{y}_p) \\
=& \sum_p (\vec{x}_p, D_p \vec{y}_p) \\
=& \sum_p (\vec{x}_{p,I}, \vec{y}_{p,I}) + \sum_p (\vec{x}_{p,B}, \vec{y}_{p,B})
\end{aligned}
$$

部分行列での重み付き内積の演算結果をスカラーの総和計算でまとめる

```
MPI_Allreduce(local, global, 1, MPI_DOUBLE, MPI_SUM,
              MPI_COMM_WORLD);
```

実習：IML++ で C++ で記述された GMRES を実装する

IML++, https://math.nist.gov/iml++/gmres.h.txt では GMRES は C++ で記述されている
Operator, Vector と Matrix クラスを簡単なもので置き換える
前処理操作 const Preconditioner &M に加法的 Schwarz 法 を用いる

- Real は double
- Vector は std::vector<double>
- Operator は structure CSRformat
- Matrix は 一次元配列 std::vector<double> を二つの添え字で参照するものとして

```
std::vector<std::voctor<double> > HH(max_iter + 1);
for (int n = 0; n < max_iter +1; n++) HH[i].reszie(max_iter + 1);
#define H(i, j) HH[(i)][(j)]
```

残差ベクトル r = b - A * x; は C++ の演算子で記述されているがこれを疎行列の GEMV ルーチンで置き換える

```
std::vector<double> r(b);
SparseGEMV(A, (-1.0), x, 1.0, r);
```

総和計算には MPI の Allreduce 関数を用いる.

```
int mpi_id, mpi_procs;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &mpi_id);
MPI_Comm_size(MPI_COMM_WORLD, &mpi_procs);

MPI_Allreduce(&tmps[0], &rhs[0], nrow, MPI_DOUBLE, MPI_SUM,
              MPI_COMM_WORLD);
```

PETSc ライブラリー：1/5

PETSc：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

- KSP：種々の前処理実装を実現する Krylov 部分空間法のパッケージ

ksp/tutorial/ex1.c の開始部分 (MPI 初期化, 行列オブジェクトの生成)

```
#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：2/5

ksp/tutorial/ex1.c の行列データの設定と Krylov 部分空間法収束条件の設定

```
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：3/5

- `MatSetVaules()` による疎行列データ設定 (送信) のあと並列プロセスでの疎行列データ構成のため`MatAssenblyBegin()` による開始操作と `MatAssenblyEnd()` による終了操作をコールする必要がある
- `PCSetType` によりどの前処理 `PCType` を利用するか指定する
- 前処理`PCType` はいくつかの手法を提供する
  `PCJACOBI`, `PCILU`, `PCSOR`, `PCGAMG`, `PCASM`
- KSPSetFromOptions は Kyrlov 部分空間法ソルバーのオプションを指定するがこれはコマンドラインからも指定できる

```
-ksp_type gmres -pc_type ilu -ksp_monitor -ksp_rtol 1.e-6
```

PETSc ライブラリー：4/5

ksp/tutorial/ex8.c は加法的 Schwarz 前処理の例を示す

```
 #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;               /* PC context */
  IS  *is, *is_local;  //
  PetscInt overlap = 1;     /* width of subdomain overlap */
  PetscInt Nsub;            /* number of subdomains */
  PetscInt m = 15, n = 17; /* mesh dimensions in x- and y- directions
  PetscInt M = 2, N = 1;    /* number of subdomains in x- and y- direct
  PetscInt i, j, Ii, J, Istart, Iend;
  PetscMPIInt size;

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

  MatCreate(PETSC_COMM_WORLD, &A);
  MatSetSizes(A, PETSC_DECIDE, PETSC_DECIDE, m * n, m * n);
  MatSetFromOptions(A);
  MatSetUp(A);
  MatGetOwnershipRange(A, &Istart, &Iend);
  for (Ii = Istart; Ii < Iend; Ii++) {
    //  MatSetValues(A, 1, &Ii, 1, &J, &v, INSERT_VALUES);
```

PETSc ライブラリー：5/5

ksp/tutorial/ex8.c は加法的 Schwarz 前処理の例を示す

```
MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY);

KSPCreate(PETSC_COMM_WORLD, &ksp);
KSPSetOperators(ksp, A, A);
KSPGetPC(ksp, &pc);
PCSetType(pc, PCASM);
PCASMSetOverlap(pc, overlap);
KSPSetFromOptions(ksp);
KSPSolve(ksp, b, x);
```

行列の分割の指定方法

- m×n の直交格子からなる計算領域を M×N の部分領域に分割するには

```
PCASMCreateSubdomains2D(m, n, M, N, 1, overlap, &Nsub,
                        &is, &is_local);
PCASMSetLocalSubdomains(pc, Nsub, is, is_local);
```

- ユーザー定義の分割を実現するには

```
PCASMSetLocalSubdomains(pc, Nsub, is, is_local);
```

整数 Nsub：部分行列の総数
ポインター IS *is：重なりのある部分行列の添字集合
ポインター IS *is_local：部分行列の内部の自由度の添字集合

PETSc ライブラリーの利用

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);
KSPSetFromOptions(ksp);
KSPSolve(ksp, b, x);
```

PETSc のアルゴリズムの選択は KSPSetFromOptions によりコマンドラインから可能

```
-ksp_type gmres -pc_type asm -ksp_monitor -ksp_rtol 1.e-12
-ksp_type gmres -pc_type gamg -ksp_monitor -ksp_rtol 1.e-12
```