筑波大学計算科学研究センター
CCS HPCサマーセミナー
「並列数値アルゴリズム I」

多田野 寛人

tadano@cs.tsukuba.ac.jp

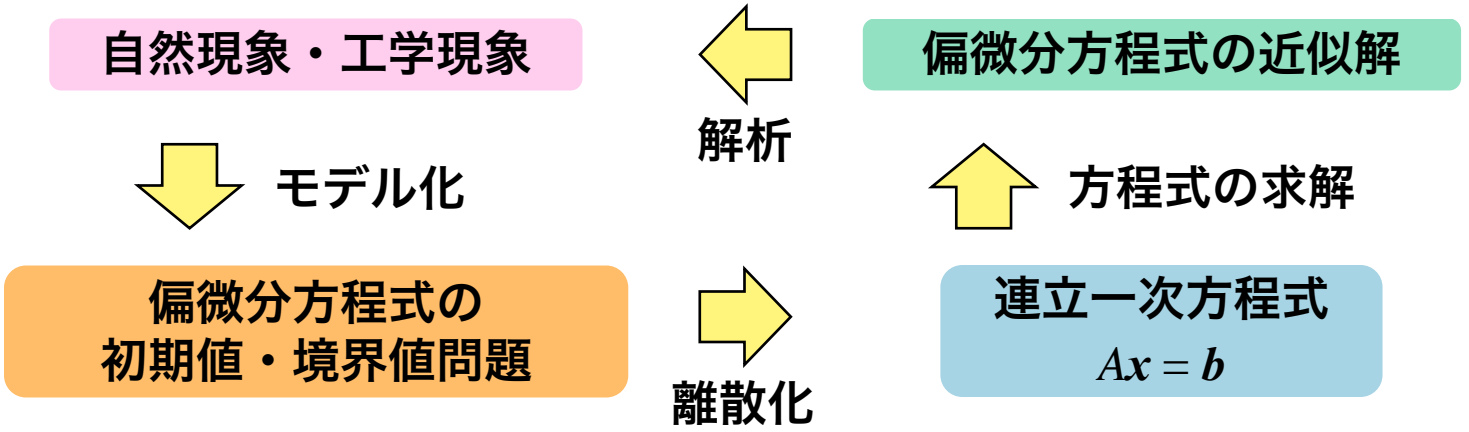
筑波大学計算科学研究センター

講義内容

- **連立一次方程式の求解法**
  - 様々な行列に対する解法
  - 疎行列の扱い方
  - 基本的な線形演算の MPI での並列化
- **複数本の右辺ベクトルをもつ連立一次方程式の解法**
  - 解法の特色
  - 効率的な計算手法と OpenMP での並列化
- **レポート課題について**

連立一次方程式の求解法

自然現象・工学現象の解析



連立一次方程式は様々な分野における数値シミュレーションで現れ,
計算時間の大部分が求解に費やされている

連立一次方程式

連立一次方程式：$A\boldsymbol{x} = \boldsymbol{b}$

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

連立一次方程式は様々な分野における数値シミュレーションで現れ,
計算時間の大部分が求解に費やされている

直接解法と反復解法

直接解法

- Gaussの消去法，LU分解法など
- 有限回の演算で必ず解くことができる
- 係数行列 A を変形するため，**非零要素数が増大**

反復解法

- 定常反復法，クリロフ部分空間反復法
- 必要な演算は係数行列 A とベクトルの積，内積など

  ⇨ 係数行列の**疎性をそのまま使える**

- 問題によっては多くの反復回数を要することがある

代表的な直接解法（Gaussの消去法）

Gaussの消去法では，連立一次方程式 $Ax = b$ を変形し，上三角行列 U をもつ連立一次方程式にした後で解ベクトル x を計算する．

連立一次方程式： $Ax = b$

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

変形

変形後の方程式： $Ux = b'$

$$\begin{bmatrix} u_{1,1} & u_{1,2} & \dots & u_{1,n} \\ & u_{2,2} & \dots & u_{2,n} \\ & & \ddots & \vdots \\ \multicolumn{3}{c}{\Large 0} & u_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_n \end{bmatrix}$$

変形後は下記の後退代入計算により，解ベクトル x の成分を計算する．

$$x_j = \frac{1}{u_{j,j}} \left(b'_j - u_{j,j+1} x_{j+1} - u_{j,j+2} x_{j+2} - \cdots - u_{j,n} x_n \right), \quad j = n, n-1, \dots, 1.$$

上三角行列をもつ方程式への変形過程の計算量は $O(n^3)$，後退代入の計算量は $O(n^2)$．

代表的な直接解法（LU分解法）

LU分解法は係数行列 A を下三角行列 L と上三角行列 U の積に分解することで，
$\boldsymbol{Ax} = \boldsymbol{b}$ を解く方法である．**係数行列だけが変形され**，右辺ベクトルは変わらない．

変形後の方程式： $LU\boldsymbol{x} = \boldsymbol{b}$

$$\begin{bmatrix} 1 & & & 0 \\ l_{2,1} & 1 & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n,2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ & u_{2,2} & \cdots & u_{2,n} \\ & & \ddots & \vdots \\ 0 & & & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

解ベクトルの計算手順

1. $\boldsymbol{y} = U\boldsymbol{x}$ とおいて $L\boldsymbol{y} = \boldsymbol{b}$ を前進代入で解き，$\boldsymbol{y}$ を求める．
2. $U\boldsymbol{x} = \boldsymbol{y}$ を後退代入で解く．

行列 A のLU分解と解ベクトルの計算量はそれぞれ，$O(n^3), O(n^2)$ である．
右辺ベクトルが複数ある場合は，LU分解法を用いた方が効率良く求解できる．

疎行列の直接解法ソフトウェア

- **MUMPS（MPI）**
  - http://mumps.enseeiht.fr/
- **SuperLU（SuperLU_DIST で MPI並列も可）**
  - http://crd-legacy.lbl.gov/~xiaoye/SuperLU/
- **Pardiso（MPI, スレッド並列も可）**
  - Intel Math Kernel Library (MKL) にも入っている
  - http://www.pardiso-project.org/
- **WSMP（MPI, スレッド並列も可）**
  - http://researcher.watson.ibm.com/researcher/view_project.php?id=1426

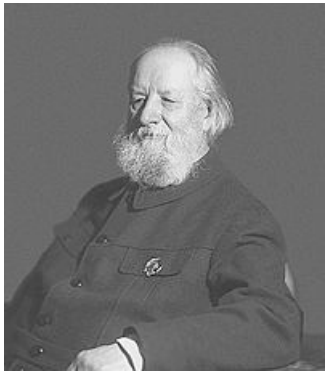
クリロフ部分空間とは？

クリロフ部分空間

行列 A と非ゼロベクトル $\boldsymbol{v}$ から生成される
ベクトル列で張られる部分空間：

$$\mathcal{K}_k(A, \boldsymbol{v}) \equiv \mathrm{span}\left(\boldsymbol{v}, A\boldsymbol{v}, \ldots, A^{k-1}\boldsymbol{v}\right)$$

を**クリロフ部分空間**という.



アレクセイ・クリロフ
（Wikipediaより）

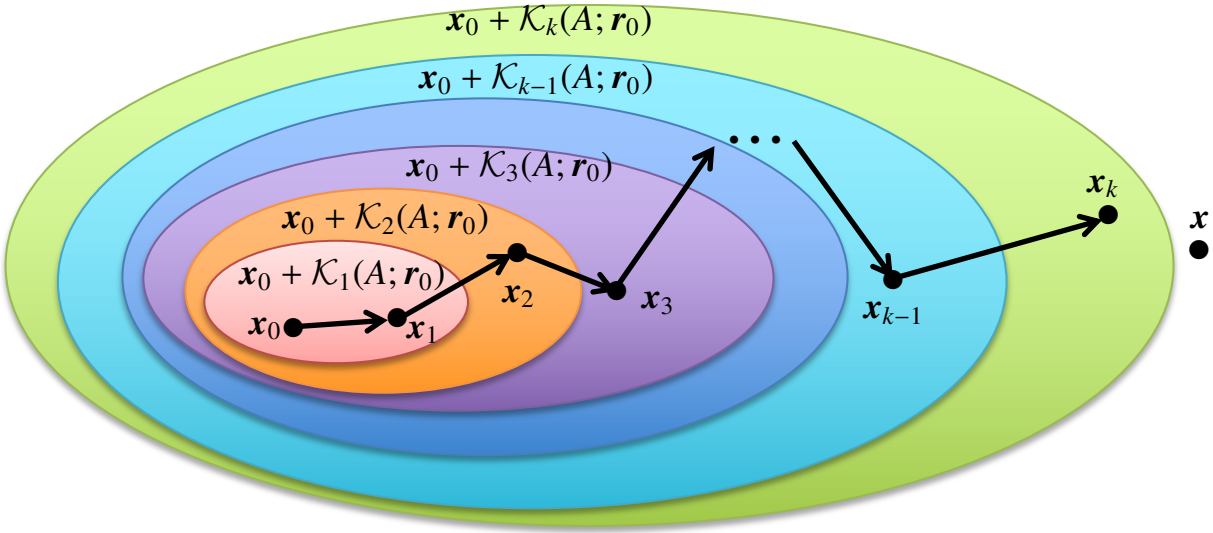
クリロフ部分空間反復法

- クリロフ部分空間を用いて連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の近似解を
  生成する方法を**クリロフ部分空間反復法**という.

- 演算の主要部は行列 A とベクトルの積で，計算量は αn^2.
  （α は係数行列の非零要素数の割合）

クリロフ部分空間反復法

- 任意の初期解 $\boldsymbol{x}_0$ を与え，反復によって近似解を更新し，真の解 $\boldsymbol{x}$ を探索
- $\boldsymbol{r}_0$ は初期残差と呼ばれ，$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$ で計算される
- 一般に真の解 $\boldsymbol{x}$ は分からないため，残差 $\boldsymbol{r}_k = \boldsymbol{b} - A\boldsymbol{x}_k$ のノルムを用いて真の解への収束をチェックする



クリロフ部分空間反復法の概念図

エルミート行列に対する解法

1. 係数行列がエルミート行列（$A = A^{\mathrm{H}}$）の場合

- **共役勾配法**（Conjugate Gradient method: **CG**法）
- **共役残差法**（Conjugate Residual method: **CR**法）
- **最小残差法**（Minimal Residual Method: **MINRES**法）

係数行列のエルミート性を使うことで
短い漸化式（計算量が少ない）の
アルゴリズムが導出できる

補足：エルミート行列

$$A = A^{\mathrm{H}} = \bar{A}^{\mathrm{T}}$$

$$(a_{ij} = \bar{a}_{ji})$$

共役勾配法（CG法）のアルゴリズム

$\boldsymbol{x}_0$ is an initial guess,

Compute $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,

Set $\boldsymbol{p}_0 = \boldsymbol{r}_0$,

For $k = 0, 1, \ldots,$ **until** $\|\boldsymbol{r}_k\|_2 \leq \varepsilon_{\mathrm{TOL}} \|\boldsymbol{b}\|_2$ **do** :

$$\boldsymbol{q}_k = A\boldsymbol{p}_k,$$ ← 行列ベクトル積

$$\alpha_k = \frac{(\boldsymbol{r}_k, \boldsymbol{r}_k)}{(\boldsymbol{p}_k, \boldsymbol{q}_k)},$$ ← 内積

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k,$$

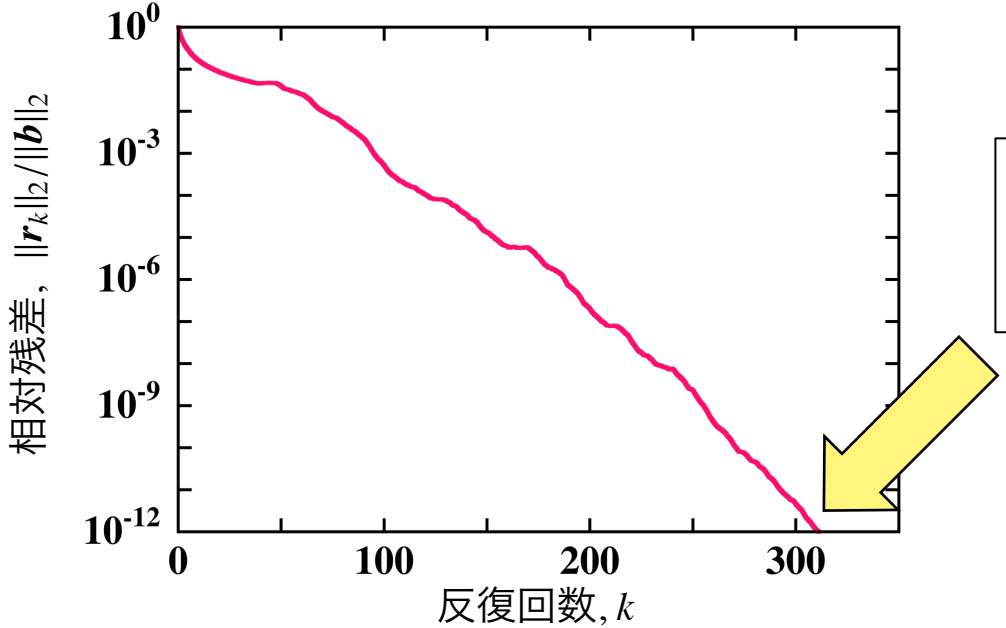
$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{q}_k,$$ ← ベクトルの定数倍と加算

$$\beta_k = \frac{(\boldsymbol{r}_{k+1}, \boldsymbol{r}_{k+1})}{(\boldsymbol{r}_k, \boldsymbol{r}_k)},$$ ← 内積

$$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k,$$ ← ベクトルの定数倍と加算

End For

CG法の相対残差履歴のイメージ



このイメージでは
$\|\boldsymbol{r}_k\|_2/\|\boldsymbol{b}\|_2 \leq 10^{-12}$
を満たしたら停止

・反復の過程で，相対残差（$\|\boldsymbol{r}_k\|_2/\|\boldsymbol{b}\|_2$）をチェックする.
・自分で定めた条件を満たしたら反復をやめ，$\boldsymbol{x}_k$ を解として採用.

非エルミート行列に対する解法

2. 係数行列が 非エルミート行列（$A \neq A^{\mathrm{H}}$）の場合

短い漸化式を用いる解法

・BiCG法　・CGS法　・BiCGSTAB法
・GPBiCG法　・BiCGSTAB(ℓ)法　・IDR(s)法　など

⇨ **計算量は少ないが，残差ノルムは単調減少しない**

長い漸化式を用いる解法

・GCR法　・GMRES法

⇨ **残差ノルムは単調減少するが，計算量が多い**

BiCG法のアルゴリズム

$\boldsymbol{x}_0$ is an initial guess,
Compute $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,
Choose $\boldsymbol{r}_0^*$ such that $(\boldsymbol{r}_0^*, \boldsymbol{r}_0) \neq 0$,
Set $\boldsymbol{p}_0 = \boldsymbol{r}_0$ and $\boldsymbol{p}_0^* = \boldsymbol{r}_0^*$,
For $k = 0, 1, \ldots,$ **until** $\|\boldsymbol{r}_k\|_2 \leq \varepsilon_{\mathrm{TOL}} \|\boldsymbol{b}\|_2$ **do**:

$$\boldsymbol{q}_k = A\boldsymbol{p}_k, \qquad \boldsymbol{q}_k^* = A^{\mathrm{H}}\boldsymbol{p}_k^*,$$

$$\alpha_k = \frac{(\boldsymbol{r}_k^*, \boldsymbol{r}_k)}{(\boldsymbol{p}_k^*, \boldsymbol{q}_k)},$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k,$$

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{q}_k, \qquad \boldsymbol{r}_{k+1}^* = \boldsymbol{r}_k^* - \bar{\alpha}_k \boldsymbol{q}_k^*,$$

$$\beta_k = \frac{(\boldsymbol{r}_{k+1}^*, \boldsymbol{r}_{k+1})}{(\boldsymbol{r}_k^*, \boldsymbol{r}_k)},$$

$$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k, \qquad \boldsymbol{p}_{k+1}^* = \boldsymbol{r}_{k+1}^* + \bar{\beta}_k \boldsymbol{p}_k^*,$$

End For

行列ベクトル積

内積

ベクトルの定数倍と加算

BiCGSTAB法のアルゴリズム

$\boldsymbol{x}_0$ is an initial guess,
Compute $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,
Choose $\boldsymbol{r}_0^*$ such that $(\boldsymbol{r}_0^*, \boldsymbol{r}_0) \neq 0$,
Set $\boldsymbol{p}_0 = \boldsymbol{r}_0$,
For $k = 0, 1, \ldots,$ **until** $\|\boldsymbol{r}_k\|_2 \leq \|\boldsymbol{b}\|_2$ **do**:

$\boldsymbol{q}_k = A\boldsymbol{p}_k,$

$$\alpha_k = \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_k)}{(\boldsymbol{r}_0^*, \boldsymbol{q}_k)},$$

$\boldsymbol{t}_k = \boldsymbol{r}_k - \alpha_k \boldsymbol{q}_k,$

$\boldsymbol{s}_k = A\boldsymbol{t}_k,$

$$\zeta_k = \frac{(\boldsymbol{s}_k, \boldsymbol{t}_k)}{(\boldsymbol{s}_k, \boldsymbol{s}_k)},$$

$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k + \zeta_k \boldsymbol{t}_k,$
$\boldsymbol{r}_{k+1} = \boldsymbol{t}_k - \zeta_k \boldsymbol{s}_k,$

$$\beta_k = \frac{\alpha_k}{\zeta_k} \cdot \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{k+1})}{(\boldsymbol{r}_0^*, \boldsymbol{r}_k)},$$

$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k(\boldsymbol{p}_k - \zeta_k \boldsymbol{q}_k),$

End For

行列ベクトル積

内積

ベクトルの定数倍と加算

GCR法のアルゴリズム

$\boldsymbol{x}_0$ is an initial guess,
Compute $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,
Set $\boldsymbol{p}_0 = \boldsymbol{r}_0$ and $\boldsymbol{q}_0 = \boldsymbol{s}_0 = A\boldsymbol{r}_0$,
For $k = 0, 1, \ldots,$ **until** $\|\boldsymbol{r}_k\|_2 \leq \varepsilon_{\mathrm{TOL}}\|\boldsymbol{b}\|_2$ **do** :

$$
\begin{aligned}
\alpha_k &= \frac{(\boldsymbol{q}_k, \boldsymbol{r}_k)}{(\boldsymbol{q}_k, \boldsymbol{q}_k)}, \\
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k, \\
\boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \alpha_k \boldsymbol{q}_k, \\
\boldsymbol{s}_{k+1} &= A\boldsymbol{r}_{k+1}, \\
\beta_{k,i} &= -\frac{(\boldsymbol{q}_i, \boldsymbol{s}_{k+1})}{(\boldsymbol{q}_i, \boldsymbol{q}_i)}, \quad (i = 0, 1, \ldots, k) \\
\boldsymbol{p}_{k+1} &= \boldsymbol{r}_{k+1} + \sum_{i=0}^{k} \beta_{k,i} \boldsymbol{p}_i, \\
\boldsymbol{q}_{k+1} &= \boldsymbol{s}_{k+1} + \sum_{i=0}^{k} \beta_{k,i} \boldsymbol{q}_i,
\end{aligned}
$$

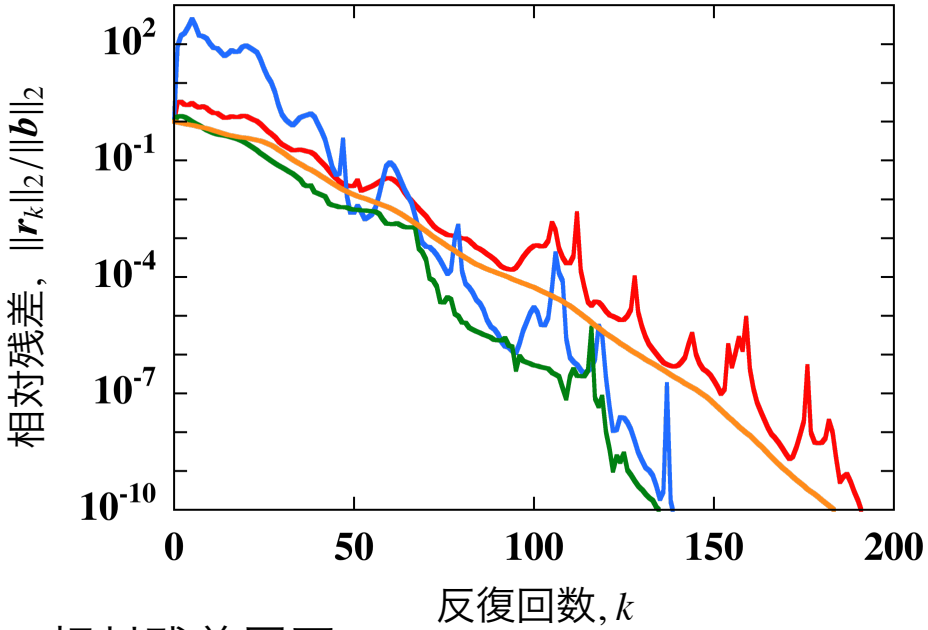
End For

- 行列ベクトル積は**1**反復あたり**1**回
- 長い漸化式を使うため多くのメモリが必要
- リスタートにより演算量とメモリ量を削減

反復法の収束特性

行列 A：poisson3Da（$n = 13{,}514$, nnz(A) = 352,762, SuiteSparse Matrix Collection）

右辺 $\boldsymbol{b}$ ：$\boldsymbol{b} = [1, 1, \ldots, 1]^{\mathrm{T}}$



反復法の相対残差履歴

■：BiCG法, ■：CGS法, ■：BiCGSTAB法, ■：GMRES法

複素対称行列に対する解法

3. 係数行列が複素対称行列（$A = A^{\mathrm{T}} \neq A^{\mathrm{H}}$）の場合

共役直交共役勾配法

（**C**onjugate **O**rthogonal **C**onjugate **G**radient: **COCG**法）

係数行列が複素対称行列の場合は,
1反復あたり**1**回の行列ベクトル積で,
かつ短い漸化式で計算ができる

補足：複素対称行列

$$A = A^{\mathrm{T}} \neq A^{\mathrm{H}}$$

$$(a_{ij} = a_{ji} \neq \bar{a}_{ji})$$

COCG法のアルゴリズム

$\boldsymbol{x}_0$ is an initial guess,

Compute $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,

Set $\boldsymbol{p}_0 = \boldsymbol{r}_0$,

For $k = 0, 1, \ldots,$ **until** $\|\boldsymbol{r}_k\|_2 \leq \varepsilon_{\mathrm{TOL}}\|\boldsymbol{b}\|_2$ **do** :

$$\boldsymbol{q}_k = A\boldsymbol{p}_k,$$ ← 行列ベクトル積

$$\alpha_k = \frac{(\bar{\boldsymbol{r}}_k, \boldsymbol{r}_k)}{(\bar{\boldsymbol{p}}_k, \boldsymbol{q}_k)},$$ ← 内積

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k,$$

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{q}_k,$$ ← ベクトルの定数倍と加算

$$\beta_k = \frac{(\bar{\boldsymbol{r}}_{k+1}, \boldsymbol{r}_{k+1})}{(\bar{\boldsymbol{r}}_k, \boldsymbol{r}_k)},$$ ← 内積

$$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k,$$ ← ベクトルの定数倍と加算

End For

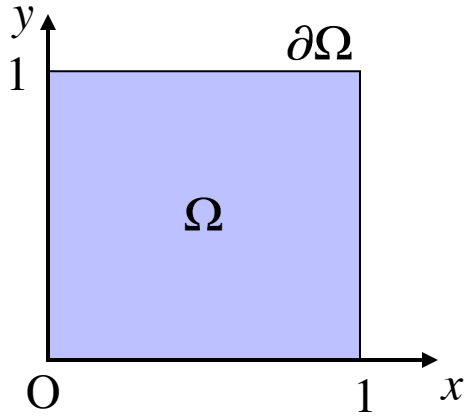
疎行列が現れる例

2次元 Poisson 問題

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} = f, & \text{in } \Omega \\ u = \bar{u}, & \text{on } \partial\Omega \end{cases}$$

f, $\bar{u}$ は既知の関数.

Ω をx, y 方向に $(M+1)$ 等分し
5点中心差分で離散化



$M \times M$次行列をもつ連立一次方程式に帰着

行列の要素数：M^4 個
非零要素の数：$5M^2 - 4M$ 個

疎行列の格納形式（CRS形式）

Compressed Row Storage (CRS) 形式
【補足】Compressed Sparse Row (CSR) と呼ぶこともあります.

$$A = \begin{bmatrix} a_{11} & 0 & a_{13} & 0 & a_{15} \\ 0 & a_{22} & 0 & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & 0 \\ 0 & a_{52} & 0 & a_{54} & a_{55} \end{bmatrix}$$

`val`: 非零要素を格納する配列

`col_ind`: 非零要素の列番号を格納

`row_ptr`: 各行の先頭の非零要素が格納されている場所を指す配列

`val:`	a_{11}	a_{13}	a_{15}	a_{22}	a_{24}	a_{25}	a_{31}	a_{32}	a_{33}	a_{43}	a_{44}	a_{52}	a_{54}	a_{55}
`col_ind:`	1	3	5	2	4	5	1	2	3	3	4	2	4	5
`row_ptr:`	1	4	7	10	12	15								

最後は非零要素数+1 の値

疎行列の格納形式（CCS形式）

Compressed Column Storage (CCS) 形式
【補足】Compressed Sparse Column (CSC) と呼ぶこともあります.

$$A = \begin{bmatrix} a_{11} & 0 & a_{13} & 0 & a_{15} \\ 0 & a_{22} & 0 & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & 0 \\ 0 & a_{52} & 0 & a_{54} & a_{55} \end{bmatrix}$$

`val`: 非零要素を格納する配列

`row_ind`: 非零要素の行番号を格納

`col_ptr`: 各列の先頭の非零要素が格納されている場所を指す配列

`val:`	a_{11}	a_{31}	a_{22}	a_{32}	a_{52}	a_{13}	a_{33}	a_{43}	a_{24}	a_{44}	a_{54}	a_{15}	a_{25}	a_{55}
`row_ind:`	1	3	2	3	5	1	3	4	2	4	5	1	2	5
`col_ptr:`	1	3	6	9	12	15								

最後は非零要素数+1 の値

CRS形式の行列ベクトル積

CRS形式では，行列 A の非零要素は行方向に連続にアクセスされる．即ち，
$\boldsymbol{y}$ の i 番目の成分 $\boldsymbol{y}_i$ は，**行列 A の i 行目とベクトル x の内積**で求められる．

行列 A とベクトル $\boldsymbol{x}$ の積 $\boldsymbol{y} = A\boldsymbol{x}$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Fortran のコード例

```
do i=1,n
   y(i) = 0.0D0
   do j=row_ptr(i), row_ptr(i+1)-1
      y(i) = y(i)+val(j)*x(col_ind(j))
   end do
end do
```

CCS形式の行列ベクトル積

CCS形式では，行列 A の非零要素は列方向に連続にアクセスされる．即ち，
行列ベクトル積 $\boldsymbol{y} = A\boldsymbol{x}$ は，**行列 A の列ベクトルの線型結合**として求められる．

行列 A とベクトル $\boldsymbol{x}$ の積 $\boldsymbol{y} = A\boldsymbol{x}$

$$\boldsymbol{y} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_n] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \sum_{i=1}^{n} \boldsymbol{a}_i x_i$$

Fortran のコード例

```
do i=1,n
   y(i) = 0.0D0
end do
do j=1,n
   do i=col_ptr(j),col_ptr(j+1)-1
      k = row_ind(i)
      y(k) = y(k) + val(i) * x(j)
   end do
end do
```

CRS形式の転置行列ベクトル積

CRS形式の転置行列ベクトル積 $z = A^{\mathrm{T}}x$

$$z = \underbrace{\begin{bmatrix} a_{1,1} & 0 & a_{3,1} & 0 & 0 \\ 0 & a_{2,2} & a_{3,2} & 0 & a_{5,2} \\ a_{1,3} & 0 & a_{3,3} & a_{4,3} & 0 \\ 0 & a_{2,4} & 0 & a_{4,4} & a_{5,4} \\ a_{1,5} & a_{2,5} & 0 & 0 & a_{5,5} \end{bmatrix}}_{A^{\mathrm{T}}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}}_{x}$$

Fortran のコード例

```
do i=1,n
   z(i) = 0.0D0
end do
do j=1,n
   do i=row_ptr(j),row_ptr(j+1)-1
      k = col_ind(j)
      z(k) = z(k) + val(i) * x(i)
   end do
end do
```

- CRS形式で格納される行列の非零要素は，行方向に連続にアクセスされる．
- この行列の転置をとると，非零要素は列方向に連絡にアクセスされることになる．
- 即ち，**CRS形式で格納された行列の転置行列ベクトル積を行うには，CCS形式の行列ベクトル積コードを実行すればよい**，ということになる．

CCS形式の転置行列ベクトル積

CCS形式の転置行列ベクトル積 $z = A^{\mathrm{T}}x$

$$z = \underbrace{\begin{bmatrix} a_{1,1} & 0 & a_{3,1} & 0 & 0 \\ 0 & a_{2,2} & a_{3,2} & 0 & a_{5,2} \\ a_{1,3} & 0 & a_{3,3} & a_{4,3} & 0 \\ 0 & a_{2,4} & 0 & a_{4,4} & a_{5,4} \\ a_{1,5} & a_{2,5} & 0 & 0 & a_{5,5} \end{bmatrix}}_{A^{\mathrm{T}}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}}_{x}$$

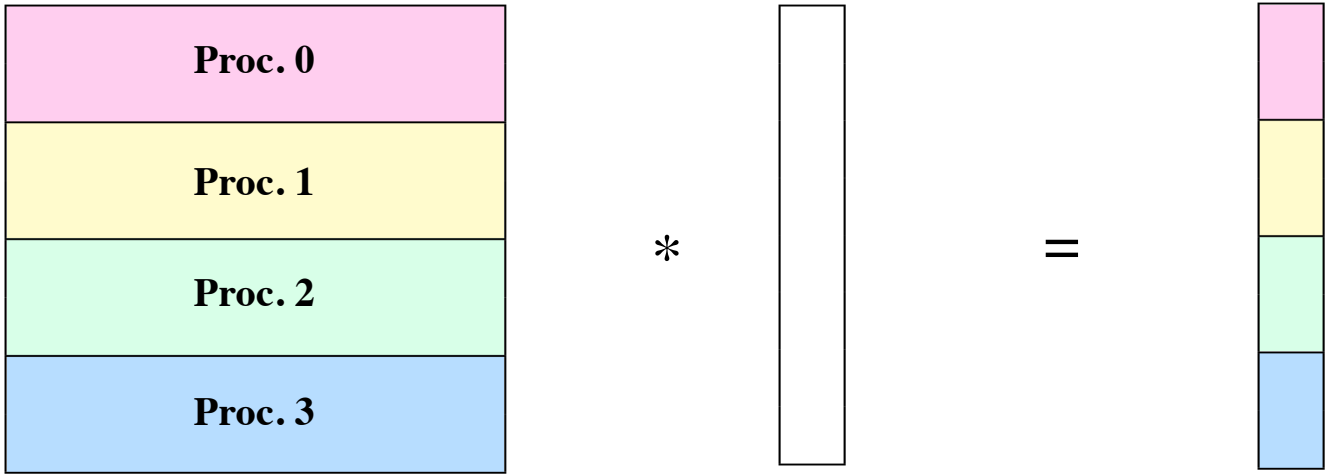
Fortran のコード例

```
do i=1,n
   z(i) = 0.0D0
   do j=row_ptr(i), row_ptr(i+1)-1
      z(i) = z(i)+val(j)*x(col_ind(j))
   end do
end do
```

- CRS形式では行列の非零要素は列方向に連続にアクセスされる，転置をとると非零要素は行方向に連絡にアクセスされることになる.
- 先程と同様に，**CCS形式で格納された行列の転置行列ベクトル積を行うには，CRS形式の行列ベクトル積コードを実行すればよい.**

行列ベクトル積の並列化

・CRS形式の場合の $y = Ax$ の計算



A

各プロセスで分散してデータを保持する

x

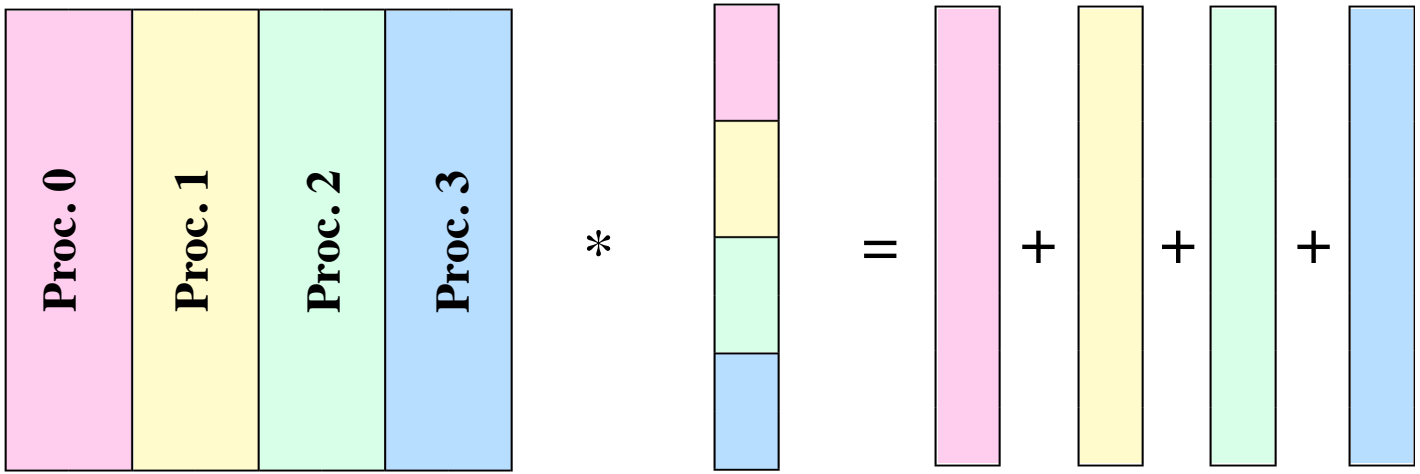
全てのプロセスで保持する

y

MPI_Gather で Proc. 0 に集める

行列ベクトル積の並列化

・CCS形式の場合の $y = Ax$ の計算



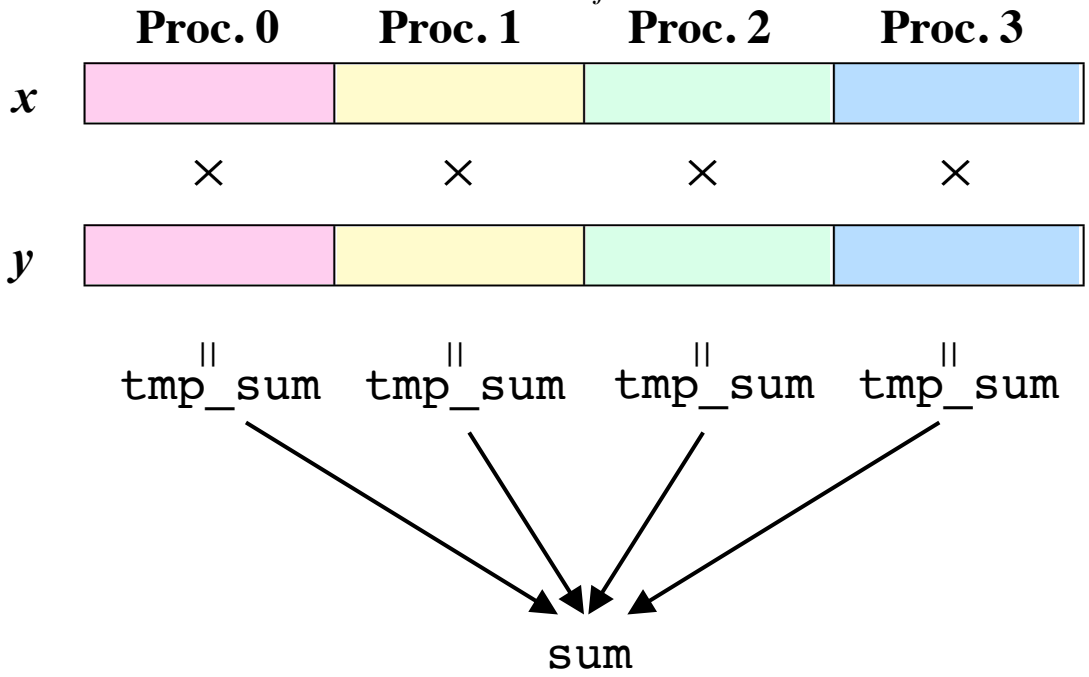
各プロセスで分散してデータを保持する

分散して保持する

MPI_Reduce でProc. 0に結果を足し合わせて送る

内積の並列化

$$(x, y) = \sum_{j=1}^{n} x_j y_j$$



MPI_Reduce で Proc.0 に集める

内積計算の MPI コード例

$$(\boldsymbol{x}, \boldsymbol{y}) = \sum_{j=1}^{n} x_j y_j$$

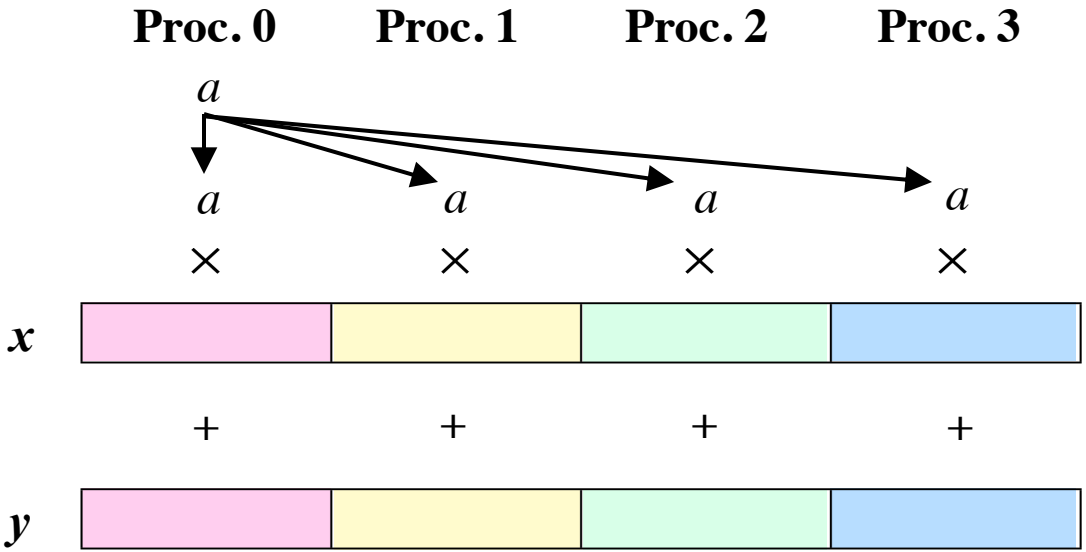
```
program main
include 'mpif.h'
...
call mpi_init(ierr)
call mpi_comm_size(mpi_comm_world, nprocs, ierr)
call mpi_comm_rank(mpi_comm_world, myrank, ierr)
...
tmp_sum = 0.0D0
do i=istart(myrank+1), iend(myrank+1)
   tmp_sum = tmp_sum + x(i) * y(i)
end do

call mpi_reduce(tmp_sum, sum, 1, mpi_double_precision,
                mpi_sum, 0, mpi_comm_world, ierr)
...
call mpi_finalize(ierr)
```

ベクトルの定数倍と加算の並列化

$y = y + ax$　（x, y：ベクトル，a：スカラー）

a を MPI_Bcast で全プロセスに送る



複数本の右辺ベクトルをもつ連立一次方程式の解法

複数右辺ベクトルをもつ方程式

右辺が s 本の連立一次方程式

$$AX = B$$

ここで，A : n次行列,

$$X = \left[\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(s)}\right], B = \left[\boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}, \ldots, \boldsymbol{b}^{(s)}\right]$$

直接法による解法

- 係数行列の完全分解（$A = LU$ など）が必要
- 完全分解できれば，s 回の前進・後退代入で OK
- 分解には多くの計算量，メモリ量が必要

反復法で s 本の方程式を効率よく解けないか？

ブロッククリロフ部分空間反復法

ブロッククリロフ部分空間反復法の種類

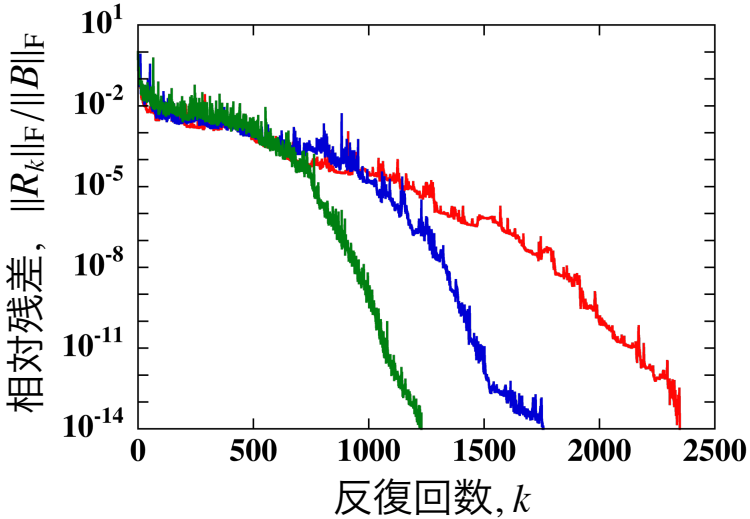
・Block BiCG法	O’Leary (1980)
・Block GMRES法	Vital (1990)
・Block QMR法	Freund (1997)
・Block BiCGSTAB法	El Guennouni (2003)
・Block BiCGGR法	Tadano (2009)

複数本の右辺ベクトルをまとめて扱うことで
効率よく解を求めることができる

ブロッククリロフ部分空間反復法

「効率よく」とはどういうことか？

1本ずつ解いた場合より少ない反復回数で残差が収束する（ことがある）



Block BiCGSTAB法の相対残差履歴.
ここで, ■：$s=1$, ■：$s=2$, ■：$s=4$.

Block CG法

$X_0 \in \mathbb{C}^{n\times s}$ is an initial guess,

Compute $R_0 = B - AX_0$,

Set $P_0 = R_0$,

For $k = 0, 1, \ldots,$ until $\|R_k\|_{\mathrm{F}} \leq \varepsilon_{\mathrm{TOL}}\|B\|_{\mathrm{F}}$ **do**:

$Q_k = AP_k$,

Solve $(P_k^{\mathrm{H}}Q_k)\alpha_k = R_k^{\mathrm{H}}R_k$ for α_k,

$X_{k+1} = X_k + P_k\alpha_k$,

$R_{k+1} = R_k - Q_k\alpha_k$,

Solve $(R_k^{\mathrm{H}}R_k)\beta_k = R_{k+1}^{\mathrm{H}}R_{k+1}$ for β_k,

$P_{k+1} = R_{k+1} + P_k\beta_k$,

End For

CG法と異なるところ

1. 行列・ベクトル積の本数が1本から s 本に増加.
2. α_k, β_k が s 次行列になった.
3. ベクトルの定数倍の計算が行列・行列積になった.

行列・ベクトル積の効率化

・行列は CRS 形式で格納されているとする.
・$Y = AX$ を計算. Y, X は n 行 s 列の配列.

```
do k=1,s     ! s についてのループが一番外側
   do i=1,n
      do j=row_ptr(i), row_ptr(i+1)-1
         Y(i,k) = Y(i,k) + val(j) * X(col_ind(j),k)
      end do
   end do
end do
```

[問題点]

・X についてメモリの連続アクセスができていない
（Fortran は行方向に連続にデータが並んでいる）
・行列データを s 回読まなければならない.

行列・ベクトル積の効率化

[解決策]
・X, Y を転置した形で保持させる

```
do i=1,n
   do j=row_ptr(i), row_ptr(i+1)-1
      do k=1,s  ! s についてのループが一番内側
         Y(k,i)= Y (k,i) + val(j) * X(k,col_ind(j))
      end do
   end do
end do
```

・X について（少なくとも s 回は）連続アクセスができる.
・行列データは1回しか読み込まない.
・Y についても連続アクセスが保たれている.

$n \times s$ 行列・$s \times s$ 行列積の計算

・行列・ベクトル積の効率化のためにベクトルを転置した.
・$n \times s$ 行列と $s \times s$ 行列の積の計算も工夫が必要.

転置

$$X_{k+1} = X_k + P_k \alpha_k \quad \Rightarrow \quad X_{k+1}^{\mathrm{T}} = X_k^{\mathrm{T}} + \alpha_k^{\mathrm{T}} P_k^{\mathrm{T}}$$

```
do j=1,n
   do i=1,s
      do k=1,s
         X(k,j) = X(k,j) + Alpha(k,i) * P(i,j)
      end do
   end do
end do
```

ベクトルと α_k を全て
転置して保持することで
連続アクセスが可能に.

Alpha は転置済みとする.

$s \times n$ 行列・$n \times s$ 行列積の計算

- α_k, β_k を求めるために必要.
- $C_k = P_k^{\mathrm{H}} Q_k$ を計算することを考える.

```
do j=1,n
   do i=1,s
      do k=1,s
         C(k,i) = C(k,i) + dconjg(P(k,j)) * Q(i,j)
      end do
   end do
end do
```

- C_k の計算も，連続メモリアクセスを保持できる.
- dconjg：複素数の共役を求める関数.

OpenMP による並列化

・共有メモリ向けの並列化インターフェース.
・既存のプログラムに数行加えるだけで並列化ができる.

```
!$OMP PARALLEL
  【 プログラム 】
!$OMP END PARALLEL
```

と書くと，スレッドが立ち上がり，スレッド毎に別々の処理ができるようになる.

（以下のコードは，**!$OMP PARALLEL** と **!$OMP END PARALLEL** で囲まれているとします．）

OpenMP による並列化

1. 行列・ベクトル積の並列化

```
!$OMP DO PRIVATE(j,k)
do i=1,n
   do j=row_ptr(i), row_ptr(i+1)-1
      do k=1,s
         Y(k,i)=Y(k,i)+val(j)*X(k,col_ind(j))
      end do
   end do
end do
!$OMP END DO
```

最初の **do** ループの前に **!$OMP DO ...** を加えるだけ.
最後の **!$OMP END DO** は省略可能.

OpenMP による並列化

2. $n \times L$行列・$L \times L$行列の積の並列化

```
!$OMP DO PRIVATE(i,k)
do j=1,n
   do i=1,s
      do k=1,s
         X(k,j) = X(k,j) + Alpha(k,i) * P(i,j)
      end do
   end do
end do
!$OMP END DO
```

一番外の **do** ループの前に **!$OMP DO ...** を加えるだけ.
最後の **!$OMP END DO** は省略可能.

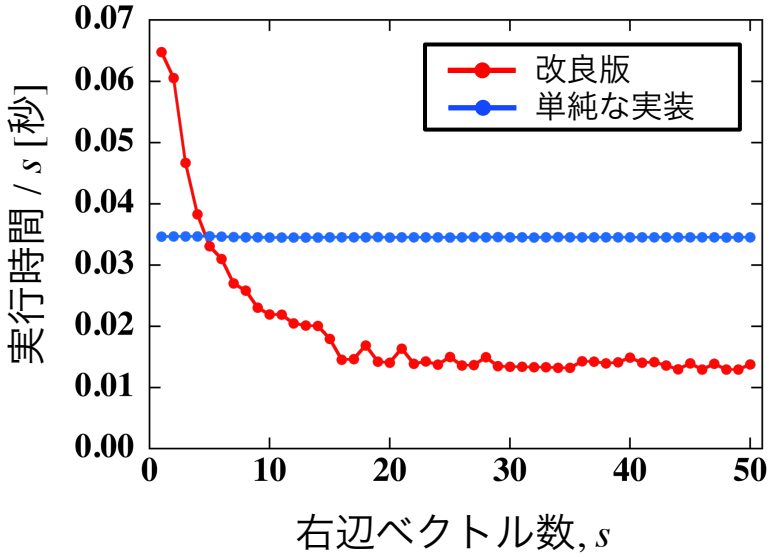
OpenMP による並列化

3. $L \times n$行列と $n \times L$行列の積の並列化

以下を実行して $C_k = P_k^{\mathrm{H}} Q_k$ を計算する.

```
!$OMP SINGLE
do j=1,L
   do i=1,L
      C(i,j) = 0.0D0
   end do
end do
!$OMP END SINGLE
!$OMP DO PRIVATE(i,k) REDUCTION(+:C)
do j=1,n
   do i=1,s
     do k=1,s
        C(k,i) = C(k,i) + dconjg(P(k,j)) * Q(i,j)
     end do
   end do
end do
!$OMP END DO
```

行列ベクトル積の性能



右辺ベクトル数の変化に対する計算時間変化（OpenMP 12並列）.

- 行列：格子量子色力学計算（QCD）で現れる行列
- 行列サイズ：1,572,864, 非ゼロ要素数：80,216,064.
- 実験環境：CPU : Intel Xeon E5-2620v3 2.4GHz × 2,
- コンパイラ：gfortran ver. 5.4, オプション：`-O3 -fopenmp`

まとめ

- 連立一次方程式の解法である
  クリロフ部分空間反復法を取り上げた.
- 疎行列に対する行列・ベクトル積の実装
  方法とその並列化について述べた.
- ブロッククリロフ部分空間反復法とコードの最適化，及び OpenMP での並列化について述べた.

レポート課題

次の行列 A と右辺ベクトル $\boldsymbol{b}$ をもつ連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を，BiCG法とBiCGSTAB法で解くプログラムを作りなさい．但し，行列 A はCRS形式で格納すること．
行列サイズ n は任意とする．

$$A = \begin{bmatrix} 2 & 1 & & & 0 \\ \gamma & 2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & \gamma & 2 & 1 \\ 0 & & & \gamma & 2 \end{bmatrix}, \quad \boldsymbol{b} = A \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix}$$

- 行列 A のパラメータ γ は $0 < \gamma < 1$ とする．反復は条件：$\|\boldsymbol{r}_k\|_2 / \|\boldsymbol{b}\|_2 \leq 10^{-12}$ を満たしたら停止すること．初期解 $\boldsymbol{x}_0$ はゼロベクトルとする．
- 複数のパラメータ γ について実験し，反復過程における相対残差 $\|\boldsymbol{r}_k\|_2 / \|\boldsymbol{b}\|_2$ をグラフにプロットするとともに考察を述べなさい．
- プログラムリストも提出すること．プログラミング言語は何を用いてもよい．