「情報システム学特別講義３」
疎行列反復解法の並列化

東京大学情報基盤センター准教授　片桐孝洋

２０１４年７月１５日（火）１４：４０－１６：１０

東京大学情報基盤センター
Information Technology Center,The University of Tokyo

講義日程
（情報システム学特別講義３）

レポートおよびコンテスト課題
（締切：
2014年8月11日（月）24時　厳守

1. ~~4月8日：ガイダンス~~
2. ~~4月15日~~
   - ~~プログラム高速化の基礎（その1）~~
3. ~~4月22日~~
   - ~~プログラム高速化の基礎（その2）~~
4. ~~5月13日~~
   - ~~MPIの基礎~~
5. ~~5月20日~~
   - ~~OpenMPの基礎~~
6. ~~5月27日~~
   - ~~Hybrid並列化技法（MPIとOpenMPの応用編）~~
7. ~~6月3日~~
   - ~~プログラム高速化の応用~~
8. ~~6月10日~~
   - ~~行列ーベクトル積の並列化~~
9. ~~6月17日~~
   - ~~べき乗法の並列化~~
10. ~~6月24日~~
    - ~~行列ー行列積の並列化~~
11. ~~7月8日~~
    - ~~LU分解の並列化~~
12. 7月15日
    - 非同期通信
    - 疎行列反復解法の並列化
13. 7月22日
    - ソフトウェア自動チューニング
14. 8月5日（補講日）
    - エクサフロップスコンピューティングに向けて

なぜ疎行列がでてくるのか？

疎行列の根拠

- 理学的、工学的な問題を、連立一次方程式などの数学上の問題に帰着する際に、疎行列が作られる
- 作られる疎行列の構造（非ゼロ要素の分布）は、以下に依存
  1. 理学的問題の場合
     - 行列演算に帰着した、数学的な問題を解く目的で、連立一次方程式などを解く
     - 非ゼロ要素の構造は、法則性はない（ことが多い）
     - 数学的に解くのが難しい（悪性の）問題であることがある

疎行列の根拠

2. 工学的問題の場合

- 工学的に意味のある問題を解く目的で、連立一次方程式などを解く
- 非ゼロ要素の構造は、規則性がある
  - 解くべき支配方程式（例：ナビエ・ストークス方程式）と、離散化手法（差分法など）に依存
  - 解くべき対象空間（物理領域）での計算メッシュ形状（三角形、六角形、・・・）に依存
- 計算メッシュの形状情報を用いて、非ゼロ要素の分布を人工的に変更できる
  （接点の順序づけの方法や、離散化手法自体を変えてしまうなど）

疎行列の例　：楕円偏微分方程式

- 領域Ω
  - 0 < x, y < 1
- 支配方程式

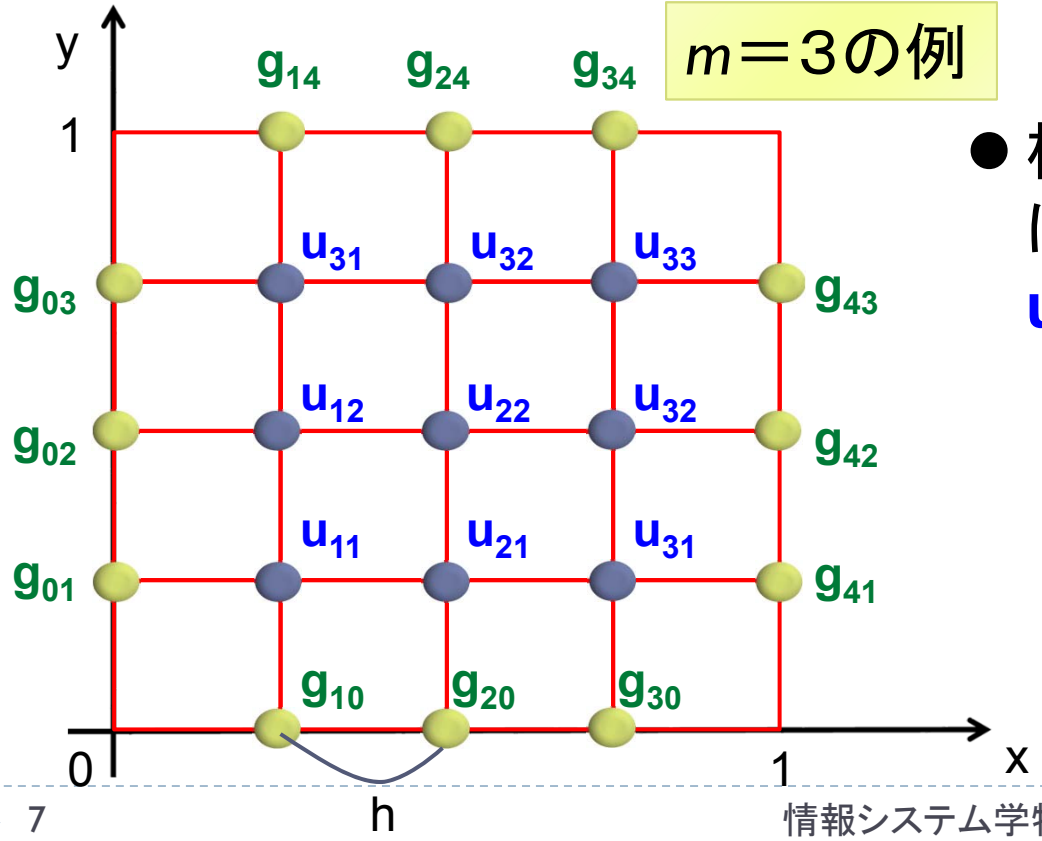
$$\frac{\partial}{\partial x}\left(-k\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(-k\frac{\partial u}{\partial y}\right)=f$$

  - 上記は、ポアソン方程式 $-\Delta u = f$とも記載される。
- 制約条件
  - 境界上で $u = g$ として、u を求める。
  - ディリクレ条件

名取亮：数値解析とその応用、コロナ社、ISBM4-339-02548-8, pp.119-125

離散格子の作成

- 領域Ωについて、x方向、y方向に、(*m*+1)等分した離散格子を作る。
- 刻み幅h　は　$h = \dfrac{1}{m+1}$



m＝3の例

- 格子点(x, y), *x*=*i h*, *y*=*j h* におけるuの値u(ih,jh)を $u_{i,j}$ と記載する。

与えられた温度：　$g_{i,j}$

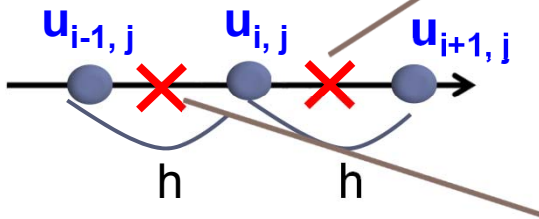
解きたい温度：　$u_{i,j}$

差分法

- ポアソン方程式を解くため、微分（偏微分）を差分で近似する
  - 差分法(finite difference method)
- 中心差分（各格子の中心の値で近似）すると　X方向の格子点の中点

$$\left[\frac{\partial}{\partial x}\left(-k\frac{\partial u}{\partial x}\right)\right]_{i,j} \approx \frac{\left(-\frac{k\partial u}{\partial x}\right)_{i+1/2,j} - \left(-\frac{k\partial u}{\partial x}\right)_{i-1/2,j}}{h}$$

一階の微分近似



$$\left[\left(-k\frac{\partial u}{\partial x}\right)\right]_{i+1/2,j} \approx -k_{i+1/2,j}\frac{u_{i+1,j}-u_{i,j}}{h}$$

$$\left[\left(-k\frac{\partial u}{\partial x}\right)\right]_{i-1/2,j} \approx -k_{i-1/2,j}\frac{u_{i,j}-u_{i-1,j}}{h}$$

差分法による近似式の導出

- 一階の差分近似を、二階の差分近似に代入すると

$$\left[\frac{\partial}{\partial x}\left(-k\frac{\partial u}{\partial x}\right)\right]_{i,j} \approx \frac{-k_{i+1/2,j}u_{i+1,j} + \left(k_{i+\frac{1}{2},j} + k_{i-\frac{1}{2},j}\right)u_{i,j} - k_{i-1/2,j}u_{i-1,j}}{h}$$

- 同様に、yに関する偏微分も、以下のように差分近似できる

$$\left[\frac{\partial}{\partial y}\left(-k\frac{\partial u}{\partial y}\right)\right]_{i,j} \approx \frac{-k_{i+1/2,j}u_{i,j+1} + \left(k_{i,j+\frac{1}{2}} + k_{i,j-\frac{1}{2}}\right)u_{i,j} - k_{i,j-1/2}u_{i,j-1}}{h}$$

ポアソン方程式の近似式

- ポアソン方程式に、二階の差分近似を代入すると、以下の方程式を得る

$$-k_{i,j-\frac{1}{2}}u_{i,j-1} -k_{i-\frac{1}{2},j}u_{i-1,j} + (k_{i+\frac{1}{2},j} + k_{i-\frac{1}{2},j} + k_{i,j-\frac{1}{2}} + k_{i,j-\frac{1}{2}})u_{i,j} - k_{i+1/2,j}u_{i+1,j} - k_{i,j+\frac{1}{2}}u_{i,j+1} = h^2 f_{i,j}$$

- ここで、定数として以下を定義した

$$k_{i,j} = k(ih, jh)$$
$$f_{i,j} = f(ih, jh)$$

疎行列の導出

- パラメタkを、k=1とすると、以下のようにポアソン方程式は簡単化される

$$-\left(\frac{\partial^2 u}{\partial x^2}-\frac{\partial^2 u}{\partial y^2}\right)=f$$

- このときの、差分近似式は、前頁の結果から

$$-u_{i,j-1}\ -u_{i-1,j}+4u_{i,j}-u_{i+1,j}\ -u_{i,j+1}=\ h^2 f_{i,j}$$

- これを、求めるべき解ベクトルu について、行列表記する。
  ここで、解ベクトルu は、以下とする。
  - uの変数の順序づけはオーダリングとよばれ、疎行列構造に影響する

$$u=(u_{11},u_{21},\cdots,u_{m1},u_{12},u_{22},\cdots,u_{m2},\cdots,u_{mm})$$

疎行列として連立一次方程式に帰着

- このとき、連立一次方程式は、以下で

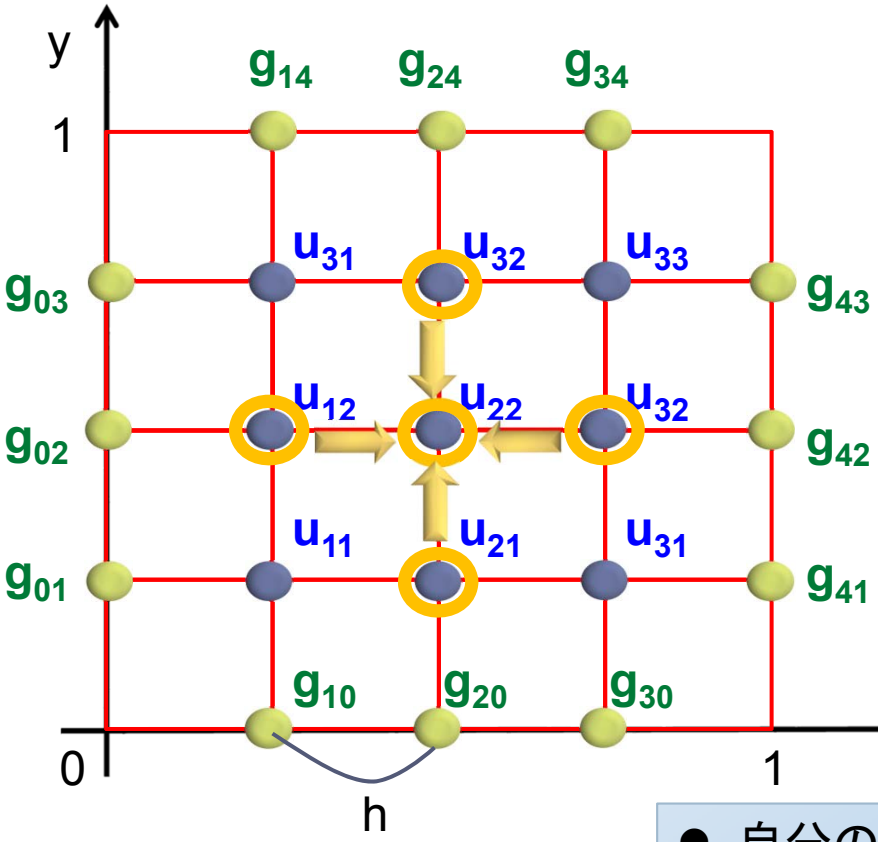
$$Au = b$$

- M=3の時の疎行列Aと、右辺ベクトルbは以下になる。

$$A = \left[\begin{array}{ccc|ccc|ccc}
4 & -1 & & -1 & & & & & \\
-1 & 4 & -1 & & -1 & & & & \\
 & -1 & 4 & & & -1 & & & \\
\hline
-1 & & & 4 & -1 & & -1 & & \\
 & -1 & & -1 & 4 & -1 & & -1 & \\
 & & -1 & & -1 & 4 & & & -1 \\
\hline
 & & & -1 & & & 4 & -1 & \\
 & & & & -1 & & -1 & 4 & -1 \\
 & & & & & -1 & & -1 & 4
\end{array}\right]$$

$$b = \begin{bmatrix}
h^2 f_{11} + g_{10} + g_{01} \\
h^2 f_{21} + g_{20} \\
h^2 f_{31} + g_{30} + g_{41} \\
h^2 f_{12} + g_{02} \\
h^2 f_{22} \\
h^2 f_{32} + g_{42} \\
h^2 f_{13} + g_{03} + g_{14} \\
h^2 f_{23} + g_{24} \\
h^2 f_{33} + g_{43} + g_{34}
\end{bmatrix}$$

計算格子と疎行列構造の関係



$m=3$の例

各格子点の対応

$$A = \begin{array}{c} \\ u_{11} \\ u_{21} \\ u_{31} \\ u_{12} \\ u_{22} \\ u_{32} \\ u_{31} \\ u_{32} \\ u_{33} \end{array} \begin{array}{c} \begin{array}{ccccccccc} u_{11} & u_{21} & u_{31} & u_{12} & u_{22} & u_{32} & u_{31} & u_{32} & u_{33} \end{array} \\ \begin{bmatrix} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & & & -1 & & & \\ -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 4 & & & -1 \\ & & & -1 & & & 4 & -1 & \\ & & & & -1 & & -1 & 4 & -1 \\ & & & & & -1 & & -1 & 4 \end{bmatrix} \end{array}$$

- 自分の要素は4倍される＝行列で対角要素は4
- 計算格子で接続される格子点は-1倍される
  ＝行列で左右の点、および一つ左右とびの点は - 1
- 例外：計算格子の四隅はつながっていないので値は無し

差分法による疎行列のまとめ

- 計算格子の構造と、差分の原理から、4点のみつながっている（自分を含めると5点）
  - ⇒一行あたり最大でも非ゼロ要素は5つ
- 要素がないところを0として扱うと、対角要素と、4つの副対角要素のみで行列が構成される
  - ⇒対角要素のデータを配列として扱うと、5本の配列のみで、疎行列が扱える
  - ⇒後述のDIA形式
- 格子上の４点（自分を含むと５点）で計算がされる
  - ステンシル行列（5点ステンシル）

疎行列データ形式と疎行列－ベクトル積（SpMV）

疎行列の表現方法

- 疎行列(Sparse Matrix)
  - 零が多い行列
- 零の値を保持すると、計算がされないので、無駄になる
- 零を除いた要素のみ保持する
  - 疎行列データ形式（Sparse Matrix Format）
- 当然、零要素がある行や列の情報を知るための
  データ構造が必要となる
- 演算性能も考慮されないといけない
  - 間接参照
  - データの連続アクセス性
  - データアクセスの局所性（キャッシュブロック化）

主な疎行列データ形式（1/2）

- COO (Coordinate)形式
  - 行および列の情報をもつ、素直な形式
- CRS (Compressed Row Storage), CSR (Compressed Sparse Row)形式
  - 行方向に非ゼロ要素を圧縮し、列情報をもつ方式
  - 実行性能の良さとともに、良く用いられている
- ELL (Ellpack)形式
  - 行あたりの非ゼロ要素数がほぼ等しいとき、効率の良い形式

主な疎行列データ形式（２/２）

- JDS (Jagged Diagonal Storage)形式
  - 列方向アクセスをできるだけ長くするように並べ替える形式
- DIA (Diagonal)形式
  - 対角行列専用の形式
  - それ以外の行列は使えない
- 混合方式
  - 疎行列を分解し、２種以上のデータ形式を利用する

疎行列データ形式と疎行列-ベクトル積

- 疎行列-ベクトル積
  (Sparse Matrix-vector Multiplication, SpMV)
  - 行列Aの疎行列データ形式を用いて、疎行列Aと密ベクトルxの行列-ベクトル積を行う処理
- 疎行列データ形式の違いで、プログラム(実装)が異なるだけでなく、実行性能が異なる
  - どの疎行列データ形式が良いかは、以下に依存
    1. 非ゼロ要素の分布状態
    2. 計算機アーキテクチャの特徴
- OpenMPでスレッド並列化も可能
  - たいていは、行ごとの演算の並列性を利用する
  - 並列性の抽出の仕方は疎行列データ形式で決まる

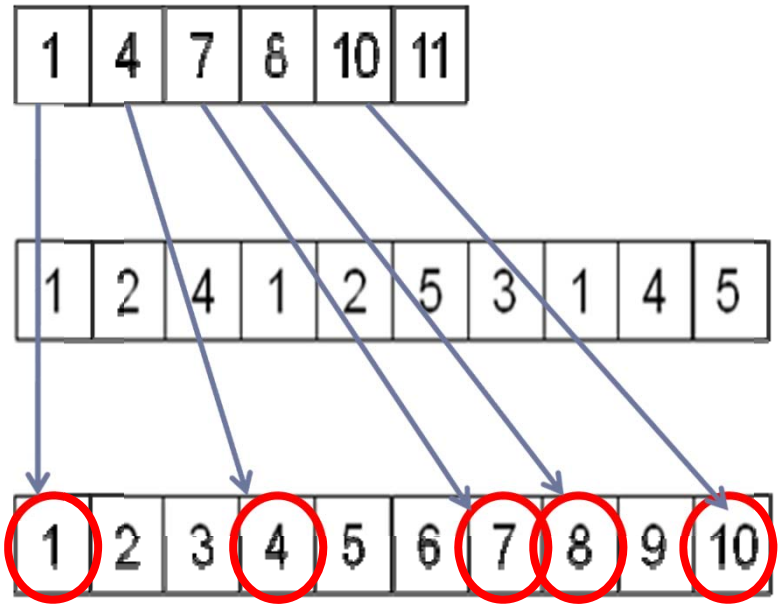
疎行列データ形式：CRS形式

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

Pointers to first row elements.

Row indexes for non-zero elements.

Values for non-zero elements.



○ 各行で最初の非零要素

CRS形式を用いたSpMV

```
!$omp parallel do private(S,  J_PTR,  I)
      DO I=1, N
        S=0.0D0
        DO J_PTR=IRP(I), IRP(I+1)-1
          S=S+VAL(J_PTR) * X(ICOL(J_PTR))
        END DO
        Y(I)=S
      END DO
!$omp end parallel do
```

行のインデックスループ

計算負荷

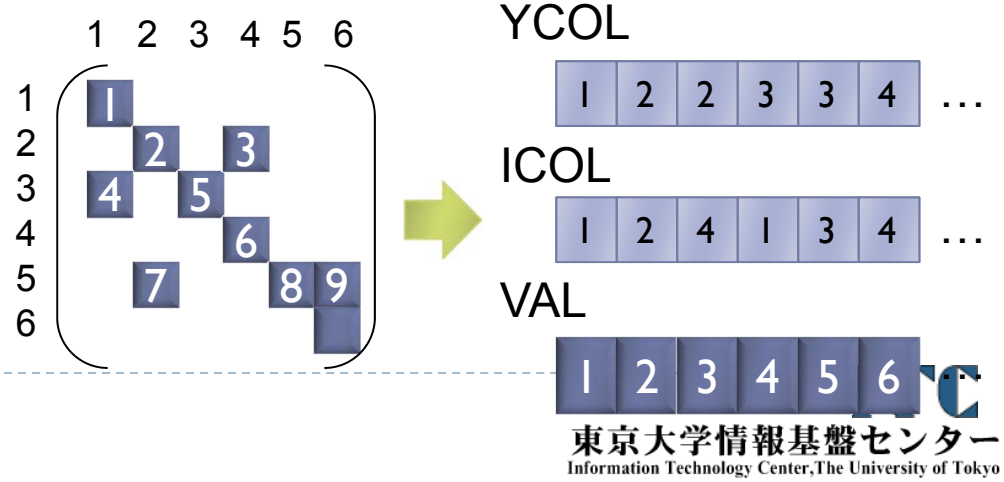
間接参照

- 疎行列の行（＝行列の次元）レベルの並列性を利用
- 各行の計算負荷は、非ゼロ要素数に異なる
  （＝単純なループ分割では、疎行列構造に依存して、負荷バランスが悪くなる）

COO形式を用いたSpMV

- 概要
  - $y = A x$ をするため、以下のデータ構造が必要
    - ICOL：右辺ベクトル x のアクセスパターン
    - YCOL：左辺ベクトル yのアクセスパターン
  - 欠点
    - 2回の間接参照が必要
  - 利点
    - 疎行列 Aの効率的な収納が可能（0要素が一切ない）
    - VAL（疎行列 Aの値のリスト）のアクセスパターンは連続
    - 最外ループのベクトル長は大変長い。ベクトル長は非ゼロ要素数（NNZ）となる。

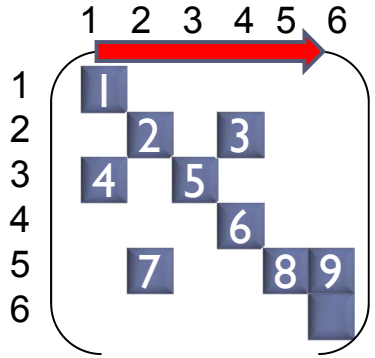
```
DO J_PTR=1, NNZ
      II = ICOL(J_PTR)
      KK = YCOL(J_PTR)
      Y(KK)=Y(KK)+VAL(J_PTR)*X(II)
END DO
```



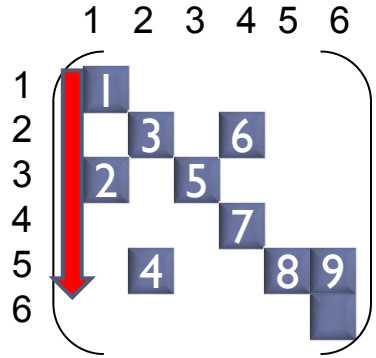
行方向圧縮と列方向圧縮

- COO形式に限らず、疎行列データ形式においては、
  - 非ゼロ要素を行方向(column-wise)圧縮していくのか
  - 非ゼロ要素を列方向(row-wise)圧縮していくのか

  の選択がある。
- 行方向圧縮と列方向圧縮のどちらが性能が良いかは、非ゼロ要素のデータ分布に依存する
  - なるべく、右辺ベクトルが連続アクセスになる方が良い



行方向圧縮



列方向圧縮

COO-行圧縮形式のSpMVのOpenMP並列化 (最外ループ)

```
<1>!$omp parallel do private(J_PTR,II,KK)
<2>       DO K=1,NUM_SMP
<3>         DO J_PTR=ISTART(K),IEND(K)
<4>           II = ICOL(J_PTR)
<5>           KK = YCOL(J_PTR)
<6>           YY(KK,K)=YY(KK,K)+VAL(J_PTR)*X(II)
<7>         END DO
<8>       ENDDO
<9>!$omp end parallel do
<10>    DO K=1,NUM_SMP
<11>      DO I=1, N
<12>        Y(I) = Y(I) + YY(I,K)
<13>      ENDDO
<14>    ENDDO
```

ループ長: NNZ

それぞれのスレッドで計算した中間結果の足しこみ部分

COO-列圧縮形式のSpMVのOpenMP並列化 (最外ループ)

```
<1>!$omp parallel do private(J_PTR,II,KK)
<2>      DO K=1,NUM_SMP
<3>        DO J_PTR=ISTART(K),IEND(K)
<4>          II = ICOL(J_PTR)
<5>          KK = XCOL(J_PTR)
<6>          YY(II,K)=YY(II,K)+VAL(J_PTR)*X(KK)
<7>        END DO
<8>      ENDDO
<9>!$omp end parallel do
<10>     DO K=1,NUM_SMP
<11>       DO I=1, N
<12>         Y(I) = Y(I) + YY(I,K)
<13>       ENDDO
<14>     ENDDO
```

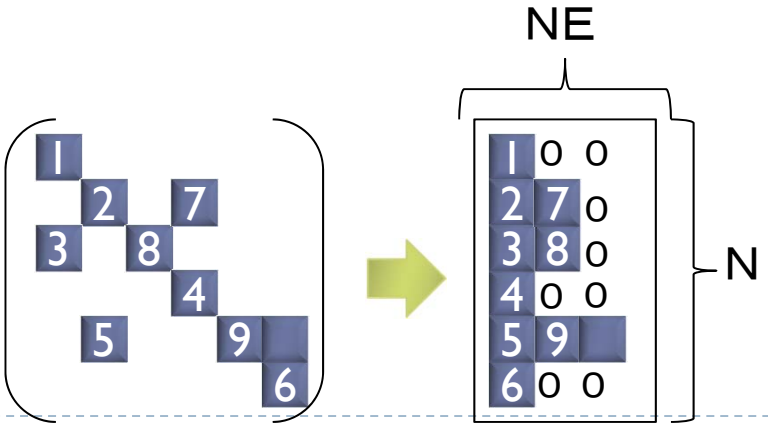
ループ長: NNZ

それぞれのスレッドで計算した中間結果の足しこみ部分

ELL形式を用いたSpMV

- 概要
  - 疎行列Aの行数をNとする。
  - 非ゼロ要素を圧縮したときの列数をNEとする。
  - 欠点
    - 圧縮時、非ゼロ要素がない場合は、強制的に「0」を入れる。
      →このことで、非ゼロ要素の圧縮効率が悪くなる。加えて、余分な演算が増える。
  - 利点
    - ループ長は、疎行列の行数Nとなり、長い。
    - 「ステンシル行列」に向く。（ステンシル行列では、形状と一致し、圧縮効率が高い）

```
J_PTR = I
DO K=1,NE
  DO I=1,N
    II = ICOL(J_PTR)
    Y(I)=Y(I)+VAL(J_PTR)*X(II)
    J_PTR = J_PTR + I
  ENDDO
ENDDO
```



ELL形式でのSpMVのOpenMP並列化

- ELL形式のSpMVは、2重ループを構成する
- そのため、OpenMPを用いた並列化も、2重ループの双方に、別々に適用できる
- 基本的には、ループ長で適用するループを決める
  - 最内ループ長はN
    疎行列の列数のため、ベクトル長は常に長い
    ←高スレッド実行に向く
  - 最外ループ長はNE
    ループ長は、疎行列の形状に依存。
    （だが、ステンシル行列だと9行など）
    ←低スレッド実行に向く

ELL-行圧縮形式　でのSpMVの OpenMP並列化 (最内 ループ)

```
<1>      DO K=1,NE
<2>        KK = N*(K-1)
<3>!$omp parallel do private(J_PTR,II)
<4>        DO I=1,N
<5>          J_PTR = KK+I
<6>          II = ICOL(J_PTR)
<7>          Y(I)=Y(I)+VAL(J_PTR)*X(II)
<8>        ENDDO
<9>!$omp end parallel do
<10>    ENDDO
```

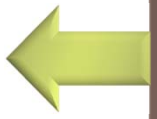
ループ長: N

利点: 各スレッドでの中間値を足しこむループが不要

ELL-行圧縮　形式でのSpMVのOpenMP並列化 (最外ループ)

```
<1>!$omp parallel do private(K,KK,I,J_PTR,II)
<2>      DO J=1, NUM_SMP
<3>        DO K=ISTART(J),IEND(J)
<4>          KK = N*(K-1)
<5>          DO I=1,N
<6>            J_PTR = KK+I
<7>            II = ICOL(J_PTR)
<8>            YY(I,J)=YY(I,J)+VAL(J_PTR)*X(II)
<9>          ENDDO
<10>       ENDDO
<11>     ENDDO
<12>!$omp end parallel do
<13>     DO K=1,NUM_SMP
<14>       DO I=1,N
<15>         Y(I) = Y(I) + YY(I,K)
<16>       ENDDO

<17>     ENDDO
```



その他の疎行列データ形式 (1/3)

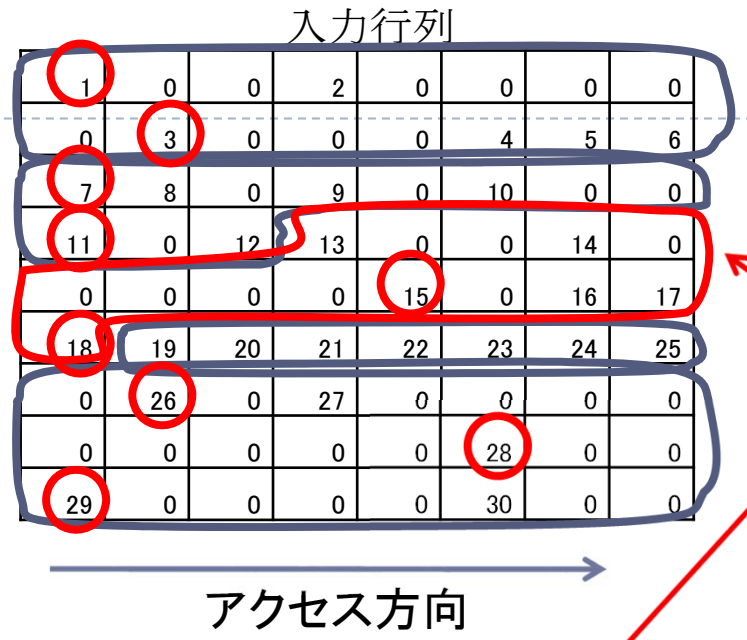
- BCRS(Blocked CRS), BCSR(Blocked CSR)
  - キャッシュブロック化を行ったCRS
  - ｍｘｍ の小行列単位で、非ゼロ要素を管理
    - 疎行列の値、行の情報の配列に加え、
      上記の小行列へのポインタ構造の配列が必要
    - ｍｘｍの小行列に要素がない場合、0を押し込む
- DIA, DS (Diagonal Storage)
  - 対角行列用。
  - M本の配列（非ゼロ要素値格納用）のみ。間接参照が無いため高性能
- JDS (Jagged Diagonal Storage)
  - ELL形式にして、零要素のところを詰める
  - ベクトル長が長くなるように、行列をソートする

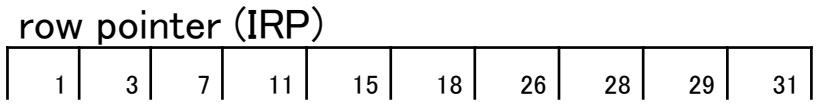
その他のSpMV実装 (2/3)

- SS(Segmented Scan)
  - 疎行列データ形式はCRS
  - 各スレッドでの計算量が均等になるように、一行当たりの非ゼロ要素を分割(もしくは融合)して、SpMVの計算をする
  - 行のつなぎの部分は、別途、足しこむ処理が入る
  - ベクトル計算機向き
  - 1行だけ非ゼロ要素数が極端に大きい行列でも高いスレッド並列性能を維持
- BSS (Branchless Segmented Scan)
  - SSの最内部に存在するIF文を取り除くデータ構造を導入
  - このことで、スカラ計算機でも高い性能を発揮する
  - 日立櫻井らにより開発され、Xabclibで採用[T.Katagiri, 2013]

Takahiro Katagiri, Takao Sakurai, Mitsuyoshi Igai, Satoshi Ohshima, Hisayasu Kuroda, Ken Naono and Kengo Nakajima: "Control Formats for Unsymmetric and Symmetric Sparse Matrix-vector Multiplications", Selected Papers of 10th International Meeting on High-Performance Computing for Computational Science (VECPAR'2012) , Springer Lecture Notes in Computer Science, Volume 7851, pp.236-248 (2013)

Segmented Scan と BSS (非対称行列のみ)

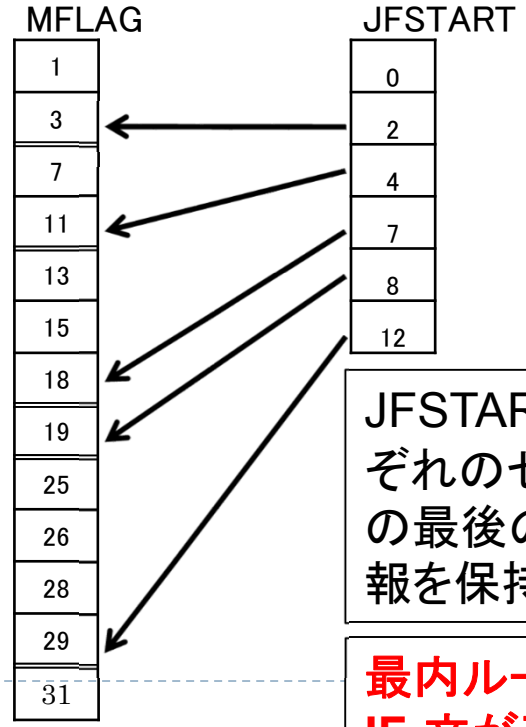




セグメント行を6要素づつ(一つの例)に分割する例。行ごとの分割ではない。

セグメントベクトル #3

Branchless Segmented Scan (BSS)

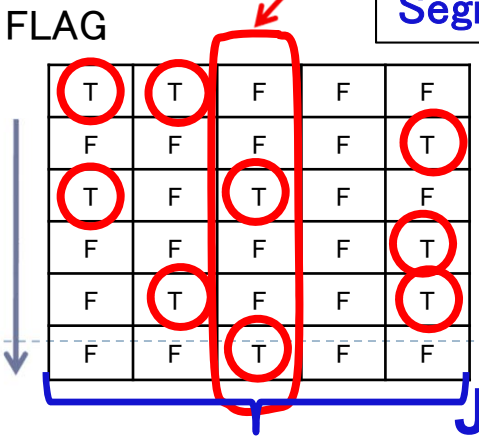


JFSTARTはそれぞれのセグメントの最後の要素情報を保持

○ それぞれの行の最初の要素

オリジナルな Segmented Scan

“T” はそれぞれの行の最初の要素



JL: セグメントベクトル数

最内ループでのIF-文が不要に！

BSSを用いたSpMVの実装

JL :セグメント長

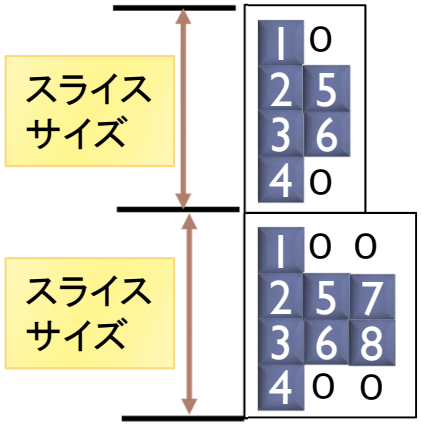
```
*  MAT*VEC product of each column
!$OMP parallel do private (K, K1, S, I)
      DO J=1,JL
        DO K=JFSTART(J-1)+1, JFSTART(J)
          K1=K+1
          S=0.0D0
          DO I=MFLAG(K), MFLAG(K1)-1
            S=VAL(I)*X(ICOL(I))+S
          END DO
          VALSS(K)=S
        END DO
      END DO
!$OMP end parallel do
*  Make spanning array
!$OMP parallel do private
      DO J=2,JL
        IF(JSFLAG(J) .EQV. .FALSE.)THEN
          SUM(J-1)=
              VALSS(JFSTART(J-1)+1)
        END IF
      END DO
!$OMP end parallel do
```

IF文を除去した行インデックス

```
*  Sum up SUM array
      DO J=JL-1,1,-1
        IF(PRESENT(J+1) .EQV. .FALSE.)THEN
          SUM(J)=SUM(J)+SUM(J+1)
        END IF
      END DO
*  Spanning sum
!$OMP parallel do
      DO J=1,JL
VALSS(JFSTART(J))=
            VALSS(JFSTART(J))+SUM(J)
      END DO
!$OMP end parallel do
*  Make output
!$OMP parallel do private (IN, JN, I)
      DO J=1,JL
        IN=JFSTART(J);   JN=JSY(J);
        DO I=JYN(J),1,-1
          Y(JN)=VALSS(IN)
          IN=IN-1;   JN=JN-1;
        END DO;  END DO;
!$OMP end parallel do
```

その他のSpMV実装 (3/3)

- Sliced ELL (Blocked ELL (BELL))
  - スライスサイズを決めて、スライスサイズごとに異なるELL形式をとる
  - ELLで0要素の混入が多い場合に有効
  - スライスサイズごとに、スレッド並列化、もしくは、MPI並列化が容易



- データ圧縮を行う形式
  - 整数Index(比ゼロ要素の行番号)を整数で保存するのではなく、前の要素からの差分で位置を表現
  - 8bit整数から、2進数に変更することで、データ圧縮する(bit操作の演算に変更する)
  - データ圧縮することでメモリアクセス量が減らせる理由から、高スレッド数での実行で高速化される

性能評価例

HITACHI SR16000/VL1

CPU	IBM POWER6 (5.0 GHz)
L1 Cache Size	64 KB/core
L2 Cache Size	4 MB/core
L3 Cache Size	32 MB/2 cores
Main Memory	1024GByte
OS	
Compiler	HITACHI Fortran90 Compiler version V02-00-/B
Compiler Option	-opt=ss -omp

ES2 (地球シミュレータセンター)

CPU	NEC SX-9/E, 8 Core/Node, 3.2GHz, 819.2GFLOPS
L1 Cache Size	
L2 Cache Size	
L3 Cache Size	
Main Memory	128GByte
OS	
Compiler	NEC SX Fortran90 Rev.404 2010/03/01
Compiler Option	-Chopt

SpMVの実装

- 自動チューニング機能付き反復解法ライブラリXabclib
  - OpenATLib ver. beta
- 疎行列データ形式
  - OpenATLibの標準データ構造CRSを利用
- SpMV スイッチ no.11
  - サブルーチン名OpenATI_DURMV
  - OpenATLibで提供

疎行列の非ゼロ要素分散の指標

- μ ：一行当たりの非ゼロ要素数の相加平均値
- σ ：一行あたりの非ゼロ要素数の標準偏差
- このとき、疎行列の非ゼロ要素の分散指標を以下のように定義する [T.Katagiri and M.Sato, 2011]

$$D_{mat} = \sigma / \mu$$

T.Katagiri and M.Sato: An Auto-tuning Method for Run-time Data Transformation for Sparse Matrix-Vector Multiplication, IPSJ SIG Technical Reports, Vol.2011-HPC-130, No.41 (2011)

テスト行列：フロリダ行列（1/2）

•行列 I

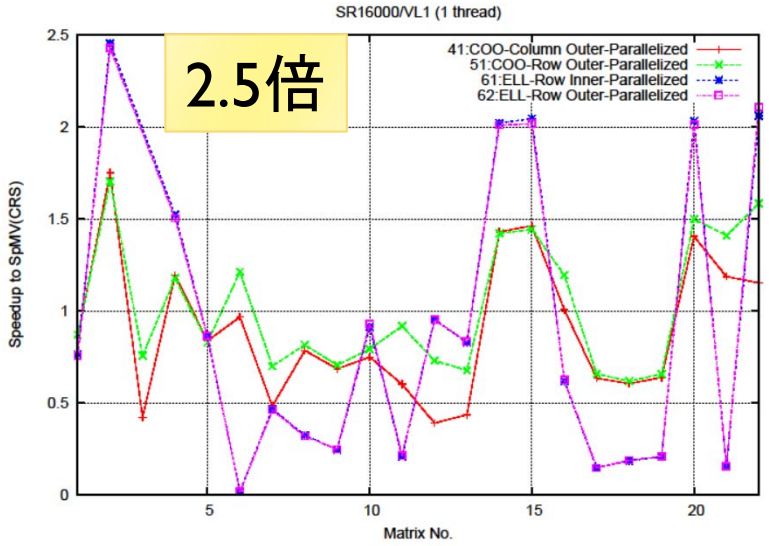
No.	Name	N	NNZ	平均 μ	分散 σ	D_{mat}	Field
1	chipcool0	20082	281150	14.00	2.69	**0.19**	2D/3D
2	chem_master1	40401	201201	4.98	0.14	**0.02**	
3	torso1	116158	8516500	73.31	419.58	**5.72**	
4	torso2	115067	1033473	8.91	0.58	**0.06**	
5	torso3	259156	4429042	17.09	4.39	**0.25**	
6	memplus	17758	126150	7.10	22.03	**3.10**	Electric circuit
7	ex19	12005	259879	21.64	12.28	**0.56**	Fluid dynamics
8	poisson3Da	13514	352762	26.10	13.76	**0.52**	
9	poisson3Db	85623	2374949	27.73	14.71	**0.53**	
10	airfoil_2d	14214	259688	18.26	3.94	**0.21**	
11	viscoplastic2	32769	381326	11.63	13.95	**1.19**	Materials

テスト行列：フロリダ行列（2/2）

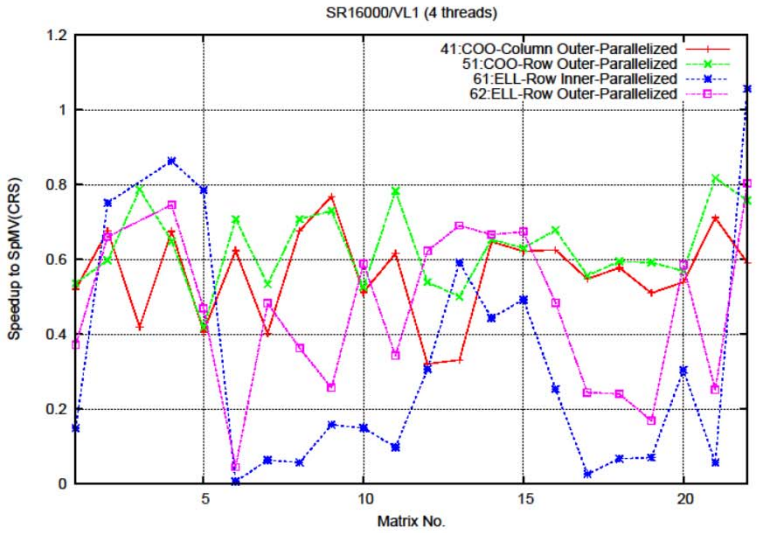
•行列II

No.	Name	N	NNZ	平均 μ	分散 σ	D_{mat}	Field
12	xenon1	48600	1181120	24.30	4.25	**0.17**	Materials
13	xenon2	157464	3866688	24.55	4.06	**0.16**	
14	wang3	26064	177168	6.79	0.43	**0.06**	Semiconductor device
15	wang4	26068	177196	6.79	0.43	**0.06**	
16	ecl32	51993	380415	7.31	3.35	**0.45**	
17	sme3Da	12504	874887	69.96	34.92	**0.49**	Structural
18	sme3Db	29067	2081063	71.59	37.06	**0.51**	
19	sme3Dc	42930	3148656	73.34	36.98	**0.50**	
20	epb1	14734	95053	6.45	0.57	**0.08**	Thermal
21	epb2	25228	175027	6.93	6.38	**0.92**	
22	epb3	84617	463625	5.47	0.54	**0.10**	

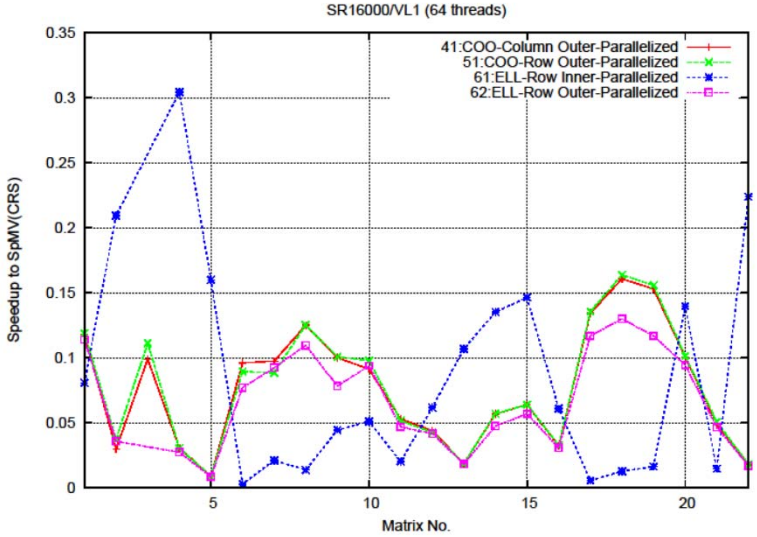
COO形式と ELL形式を用いたときの速度向上率 (HITACHI SR16000/VL1)



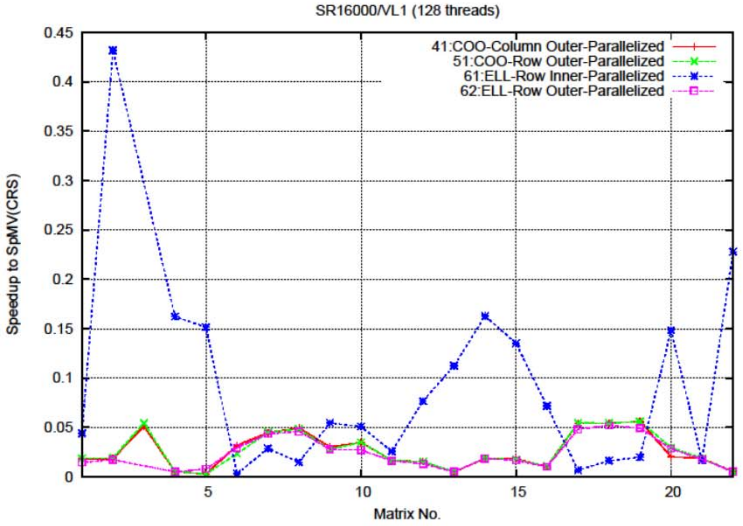
(a) 1 Thread



(b) 4 Threads

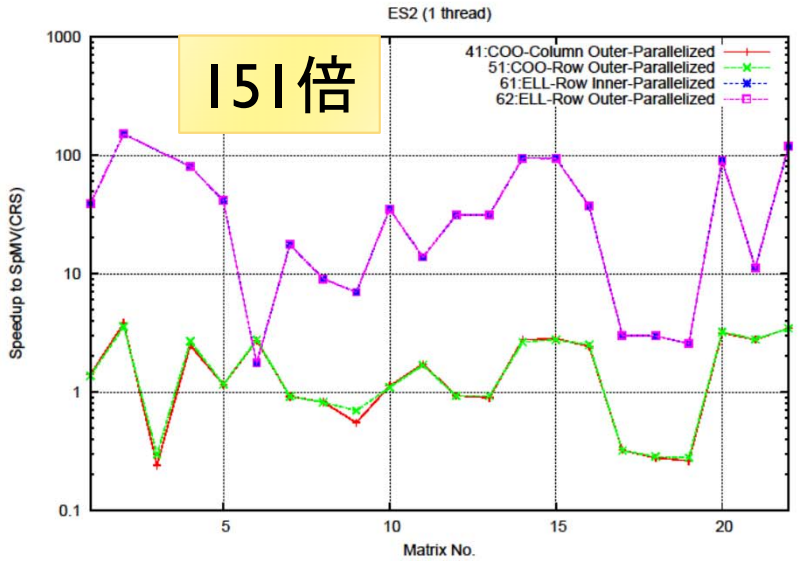


(c) 64 Threads

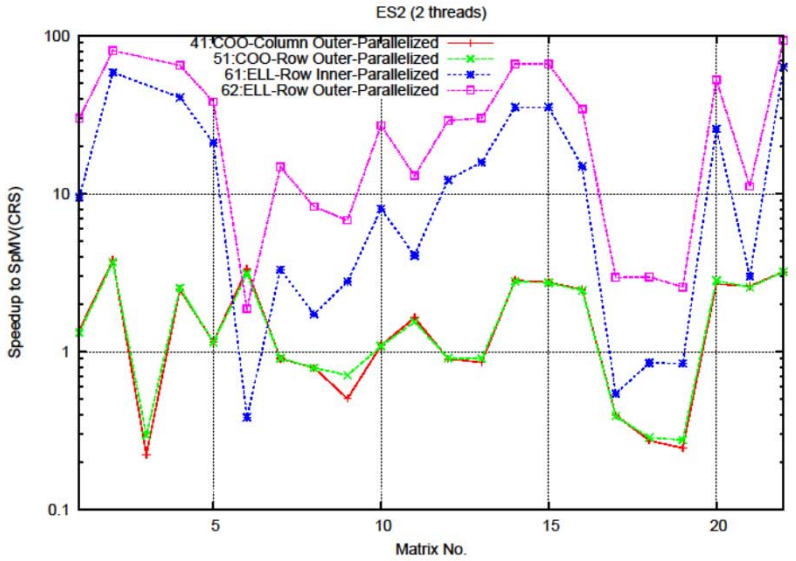


(d) 128 Threads

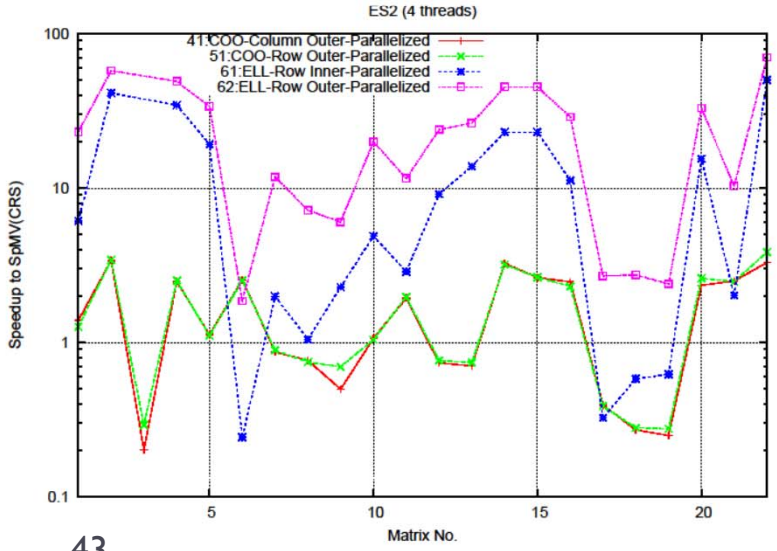
COO形式と ELL形式を用いたときの速度向上率 (ES2)



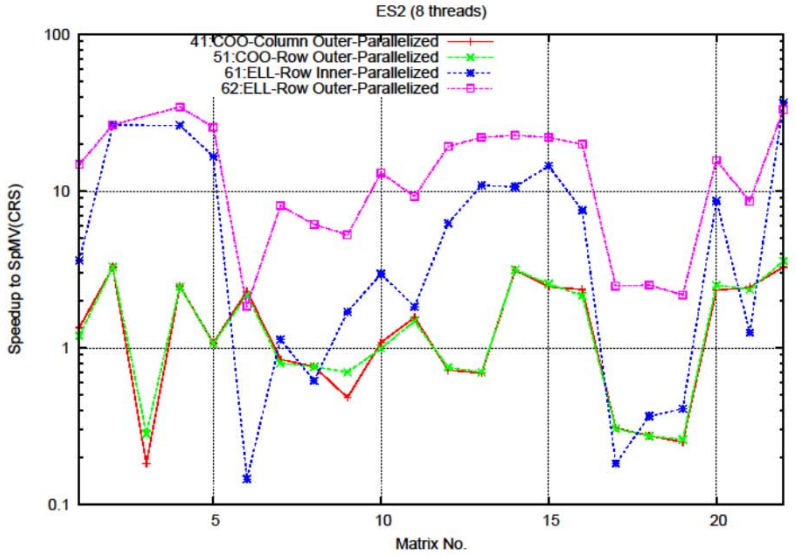
(a) 1 Thread



(b) 2 Threads



(c) 4 Threads

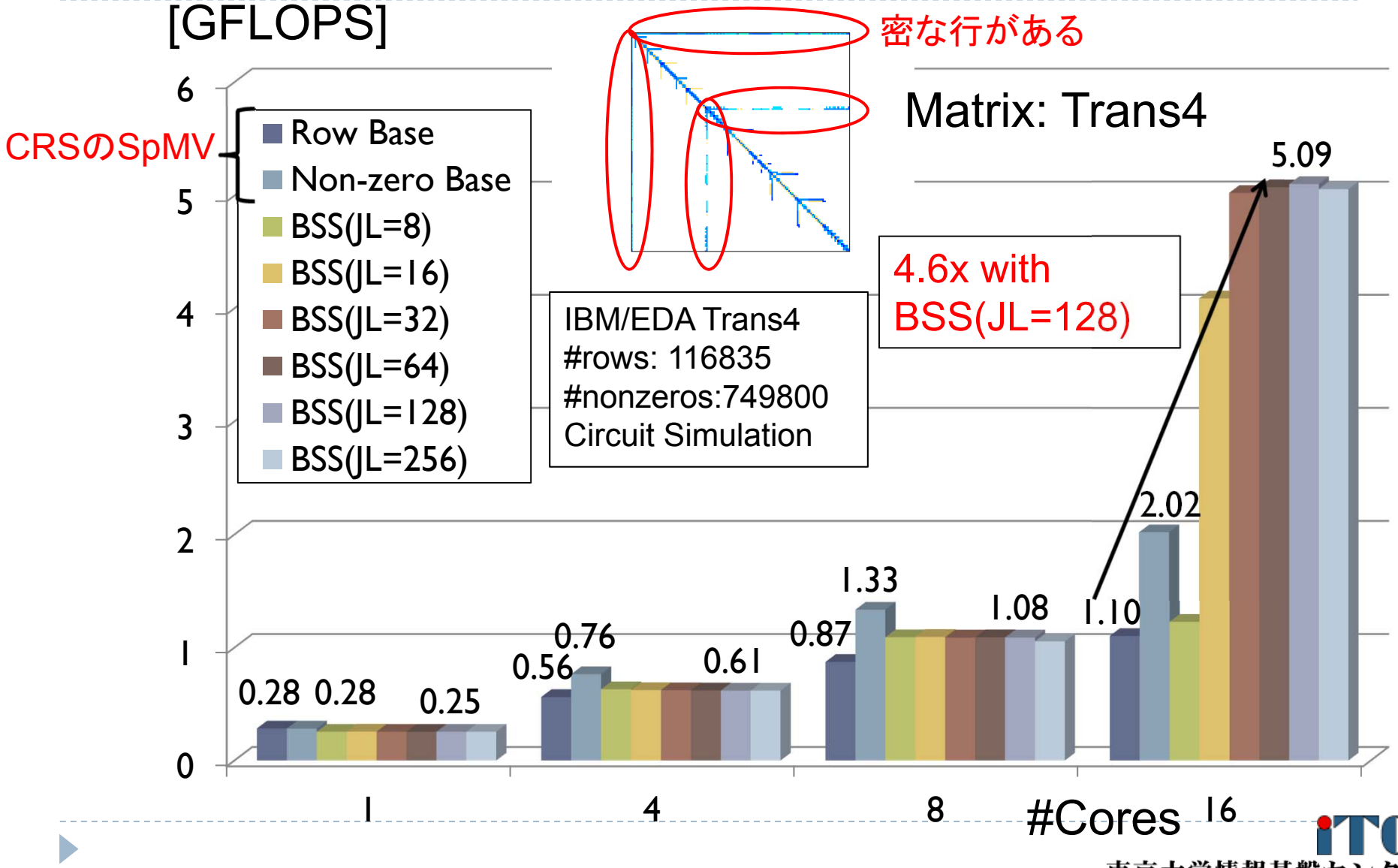


(d) 8 Threads

典型的な性能

- ベクトル計算機では、ELL形式が速い
  - 約100倍以上、CRS形式より速い
  - 一行あたりの非ゼロ要素の分布のばらつきが多い場合は、圧縮効率が悪くなり、メモリが取れないか、効果がなくなるのでダメ
- スレッド並列数が増えていくと、CRS形式が有利
  - ELL形式では、並列数が少ないか同期が多い
  - CRS形式だと、並列性が高く、並列数が増えるとキャッシュヒット率が上がる場合がある

BSSの性能
(T2Kオープンスパコン、AMD Quad Core Opteron)



分散メモリ版（MPI版）のSpMVの問題

- 通信時間について以下の考慮が必要
  1. 通信回数の最小化
     - 通信レイテンシを最小化
     - 通信する相手のプロセスは、非ゼロ要素の分散に依存する。
     - 何もMPI_allreduceを使うと、無駄な通信になる
     - 通信相手のプロセスを調べてリストにしておき、Send, Recv (もしくは、Isend, Irecv)で実装する
     - さらに、同一プロセスに送るメッセージをくっつけて1つのメッセージにする
  2. 通信量の最小化
     - 転送時間を最小化
     - 送られるメッセージの総量が少なくなるように、データ分散をする
- 上記を考慮し、疎行列をデータ分散する
- 通常、SpMVでは通信量が少ないため、通信回数の最小化が通信時間の削減につながることが多い

反復解法と疎行列 - ベクトル積（SpMV）

疎行列の零要素を増やさない解法 ー反復解法

- 疎行列に対して、LU分解法などの直接解法を適用すると、処理の過程で、0要素に0でない要素が入る
  - フィルイン(fill-in)と呼ばれる
  - いずれ密行列になって、メモリ量と演算量の利点が無くなる
- そこで、疎行列構造を変化させないで、解を近似的に解くことで、メモリ量の削減と、演算量の削減を狙う
  - これが、反復解法(Iteration Method)

SpMVと反復解法

- 反復解法では、できるだけ簡単な疎行列操作のみでアルゴリズムが構築できるとよい
- 特に、疎行列 - ベクトル積(SpMV)を主演算として、構築できるとよい
- SpMVのみで構築できる疎行列反復解法は、現在も多数、研究されている
  - 一例）クリロフ部分空間法の系列
  - 工学的な問題では、アルゴリズムの違いよりも、工学的な対象の物理特性を考慮した前処理(Preconditioner)の実装が重要
    - 前処理：
      より速く（少ない反復回数で）、解に収束させるため、
      反復に入る前におこなう処理のこと

反復解法の例：CG法

- ここでは、CG法 (Conjugate Gradient Method)を紹介する。
- CG法の原理
  - 連立一次方程式　$Ax = b$ の解　$\hat{x}$は、以下の関数を最小にすることと同じ。
    - ただし、行列Aは対称行列でかつ正定値行列であること。
      - 対称行列だが正定値行列でない場合は、最小ではないが、局所解に収束する

$$f(x) = \frac{1}{2}(x, Ax) - (x, b)$$

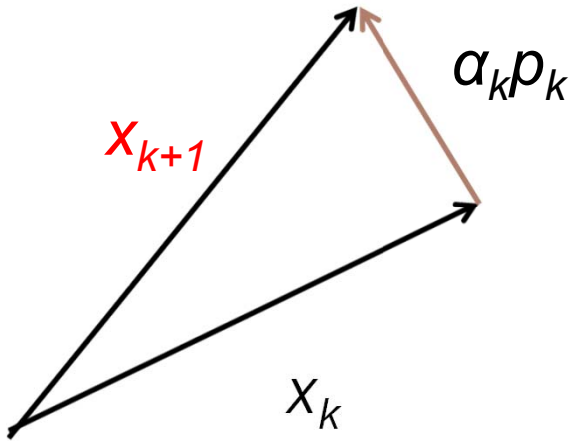
- CG法では、まず、任意の初期ベクトルx_0 から始め、$f(x)$ の最小値を探索する。

名取亮：数値解析とその応用、コロナ社、ISBM4-339-02548-8, pp.27-35

反復解法の例：CG法

- いま、k反復時において、解ベクトルの近似値x_kと、探索方向 p_k が決まったとする。
- このとき、次の反復k+1の近似値は

$$x_{k+1} = x_k + \alpha_k p_k$$



$f(x)$を最小化するパラメタα_kの決定

- この近似値が、関数$f(x_{k+1})$を最小となるように、探索方向のベクトル長を決めるパラメタα_kを決定する。
- 関数$f(x_{k+1})$を書き下し、$f(x_k)$ を用いて表現すると

$$f(x_{k+1}) = \frac{1}{2}\alpha_k{}^2(p_k, Ap_k) - \alpha_k(p_k, r_k) + f(x_k)$$

- したがって、関数$f(x_{k+1})$を最小化する α_kは

より

$$df(x_{k+1})/d\alpha_k = 0$$

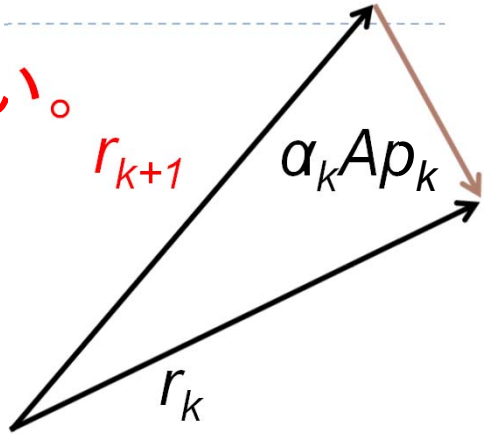
$$\alpha_k = \frac{(p_k, r_k)}{(p_k, Ap_k)}$$

ただし、残差ベクトルを $r_k = b - Ax_k$ と置いた。

残差式の計算

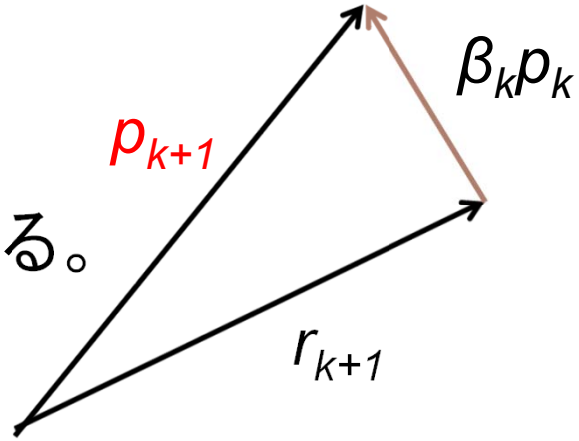
- まだ、探索方向ベクトルp_kが決まっていない。
- 第k反復時の残差は

$$r_{k+1} = r_k - \alpha_k A p_k$$



となる。残差の探索方向は、以下の式で決める。

$$p_{k+1} = r_{k+1} + \beta_k p_k$$



- ただし、最初の反復は、初期残差で決める。

$$p_0 = r_0$$

$f(x)$と同値な残差関数の定義

- いま、誤差関数を、以下のように定義する

$$e(r) = (r, A^{-1}r)$$

これは、$f(x)$ で表現すると

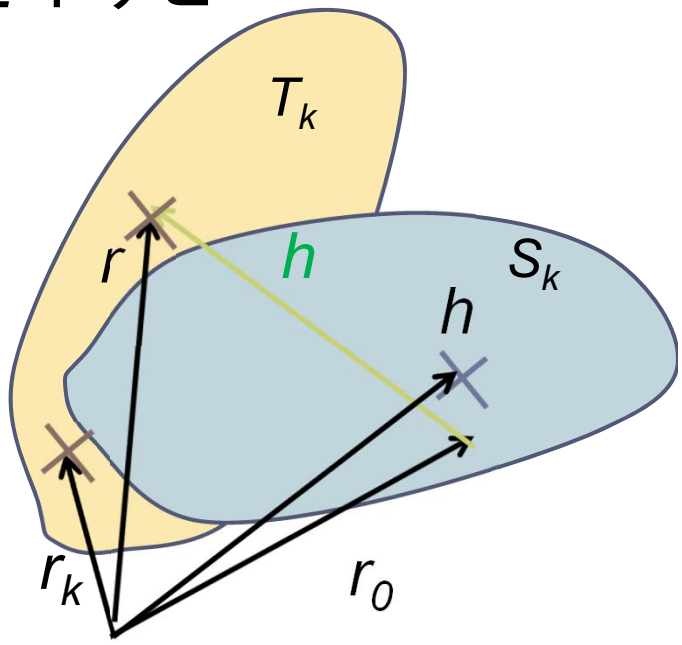
$$e(r) = 2f(x) + (b, A^{-1}b)$$

となるので、$f(x)$ を最小化すると、$e(r)$も最小化されるので、$e(r)$の最小化を考えればよい。

$f(x)$と同値な残差関数の定義

- いま、r_k を、各反復の探索方向で書き下すと

$$r_k = r_0 + \sum_{l=1}^{k} \alpha_l{}^{(k)} A^l \ r_0$$



ここで

$S_k = Span\{Ar_0, A^2 r_0, \ldots, A^k r_0\}$

: SpMVが張る空間＝クリロフ部分空間

$T_k = \{r \in R^n; r = r_0 + h, h \in S_k\}$

: 残差ベクトルが張る部分空間

とすると $r_k \in T_k$ となる(∵上記のhとr_kの赤線はS_kに含まれる)。

すなわち、SpMVのみで構成される部分空間S_kから
取る任意のベクトルhで、k反復時の残差r_kが表現できる

k反復時の残差関数の最小化

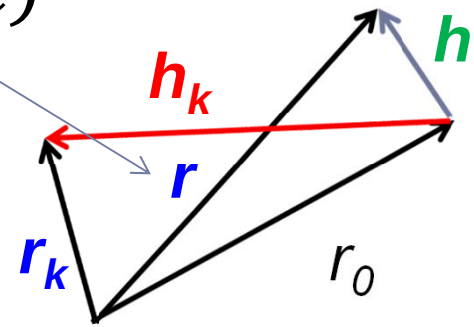
- β_kは、以下の式を満たすように決める。

$$e(r_k) = \min_{r \in T_k} e(r)$$

残差の部分空間Tkから取ったベクトルrを用いて、k反復時の誤差関数を最小化する

- これは、以下の式と等価なので、これを最小化する。

$$e(r_0 + h_k) = \min_{h \in S_k} e(r_0 + h)$$



上記となる解h_k を求める。ここで、

$$h_k = r_k - r_0$$

k反復時の残差ベクトルを最小化するβ_kの決定

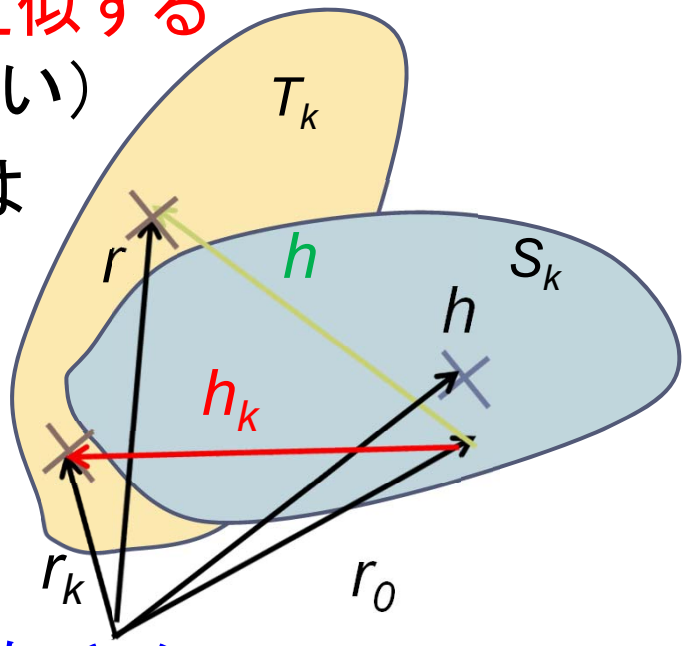
- 初期残差ベクトルをr_0 とし、$-r_0$をよく近似するh_kを見つけたい（$\because r_k$が精度よく求めたい）
- この時、最小化すべき誤差関数の式は

$$e(r_0 + h) = \left(r_0 + h, A^{-1}(r_0 + h)\right)$$
$$= (r_0 + h, A^{-1}r_0) + (r_0 + h, A^{-1}h)$$

なので、上式を最小化するには：

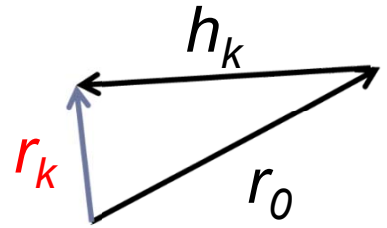
$$(r_o + h_k, A^{-1}h) = 0, \forall h \in S_k$$

r_k と hは互いに共役



- 以上のようにでh_kを決めると、誤差関数$e(r_k)$は最小化される。なお、r_kは、h_kの定義より以下となる。

$$r_k = r_0 + h_k$$



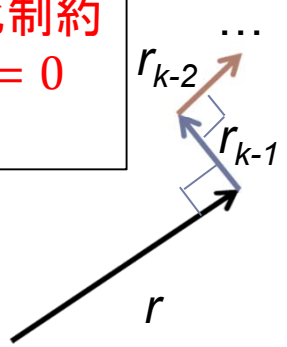
- 次に、r_k とr_0の関係を調べる。

残差ベクトルの直交性

$$r = r_0 + h,$$
$$Ar = Ar_0 + Ah \in S_k$$

- 任意の $r \in T_{k-1}$ に対して $h = Ar$ は、S_k に含まれる。
- いま、$(r_k, r) = 0, \forall r \in T_{k-1}$
  - k-1反復時の r とk反復時の r_k は直交している

誤差関数最小化制約
$(r_o + h_k, A^{-1}h) = 0$
$\Rightarrow (r_k, r) = 0$



- $l < k$ とすると、$r_l \in T_l \subseteq T_{k-1}$ ($\because h = Ar \in S_k$)
  - なので $(r_k, r_l) = 0, l \neq k$
  - つまり、k反復より前の残差ベクトルは、すべて直交している

⇒残差ベクトルは、高々、行列の次元nで必ず0になる⇒解へ収束する

- $l < k$ とするとき、以下を考える

$$\begin{aligned}(p_k, Ap_l) = (Ap_k, p_l) &= {\alpha_k}^{-1}(r_k - r_{k+1}, p_l) \\ &= {\alpha_k}^{-1}(r_k - r_{k+1}, r_l + \beta_{l-1}p_{l-1}) \\ &= \frac{\beta_{l-1}}{\alpha_k}(r_k - r_{k+1}, p_{l-1}) \\ &= {\alpha_k}^{-1}\left(\prod_{i=0}^{l-1}\beta_i\right)(r_k - r_{k+1}, r_0) = 0\end{aligned}$$

β_kの導出

- そこで、探索方向 p_{k+1} を、Ap_kと直交する方向に取ると

$$(p_{k+1}, Ap_k) = (r_{k+1} + \beta_k p_k, Ap_k)$$
$$= (r_{k+1}, Ap_k) + (\beta_k p_k, Ap_k) = 0$$

これから、

$$\beta_k = -\frac{(r_{k+1}, Ap_k)}{(p_k, Ap_k)}$$

CG法のアルゴリズム（前処理なし、素朴版）

1. 適当な初期ベクトルx_0を作る
2. 初期残差ベクトルと探索方向を決める
   $: p_0 = r_0 = b - Ax_0$
3. k=0, 1, …, MAX_ITER まで繰りかえす
4. $\alpha_k = \frac{(p_k, r_k)}{(p_k, Ap_k)}$
5. $x_{k+1} = x_k + \alpha_k p_k$
6. $r_{k+1} = r_k - \alpha_k Ap_k$
7. $\beta_k = -\frac{(r_{k+1}, Ap_k)}{(p_k, Ap_k)}$
8. $p_{k+1} = r_{k+1} + \beta_k p_k$
9. if (r_{k+1} が十分に小さい) break;

疎行列-ベクトル積(SpMV)

上記のSpMVの結果を使うのでSpMVはしない

その他、内積は3回

ＣＧ法のまとめ (1/2)

- SpMVを主演算（最も時間がかかる処理）として構成されるアルゴリズムである
  - そのほかに、内積、ベクトルとベクトルの引き算、が必要
  - 以上から、主な並列化の処理はSpMVの並列化になる
  - 超並列実行では、同期処理削減のため、内積計算の回数を減らすことが必要
    - CACG(Communication Avoiding CG)法として、研究されている
    - 高橋秀俊版のCG法というものがあり、これを使うと、少し計算量が少ない（内積回数は3回から2回に減る）。

ＣＧ法のまとめ (2/2)

- 実用では、収束回数を減らすための前処理が実装されている
  - 前処理付CG法(Preconditioned CG, PCG)
  - 前処理は、不完全コレスキー分解(Incomplete Cholesky CG, ICCG法)が使われることが多い
- PCGを並列化する際、SpMVよりも前処理の時間がほとんどを占めることが多い
  - 前処理の並列性は、疎行列構造に依存
  - 前処理の時間は、主反復に入る前の分解と主反復中の処理の2つ
  - 前処理のため、CG法の主反復ループに前進代入と後退代入が必要になる

レポート課題（その１）

1. [L10] 疎行列 - ベクトル積(SpMV)を実現し、性能評価せよ。
2. [L20] 複数の疎行列データ形式によるSpMVを実装し、性能を比較せよ。
3. [L20] 複数のSpMVをスレッド並列化し、性能評価せよ。
4. [L10] 高橋秀俊版CG法を調べて実装し、通常版と性能を比べ評価せよ。

問題のレベルに関する記述：
- L00: きわめて簡単な問題。
- L10: ちょっと考えればわかる問題。
- L20: 標準的な問題。
- L30: 数時間程度必要とする問題。
- L40: 数週間程度必要とする問題。複雑な実装を必要とする。
- L50: 数か月程度必要とする問題。未解決問題を含む。

※L40以上は、論文を出版するに値する問題。

レポート課題（その２）

5. [L30] SpMVを分散並列化（MPI並列化）せよ。
6. [L15] CG法を、SpMVを用いて実装し、性能評価せよ。
7. [L15] 前処理付CG法(ICCG法)の原理を説明し、実装して性能評価せよ。
8. [L25] CG法をSpMVを用い、かつ、MPI並列化せよ。また、性能評価を行え。
9. [L30] 複数のSpMVの実装を用いてCG法を実装し、かつ、ハイブリッドMPI並列化せよ。また、性能を評価せよ。