省メモリ型の大規模連立方程式解法について

1　依頼内容

1.1　依頼内容の概略

低次元電子系におけるスペクトルおよび励起状態の性質を調べるために，励起エネルギーごとにいわゆる sorrection vector を求める．4×10^9 のオーダーの次元を持つ大規模行列の連立方程式を共役勾配（CG）法で解く．

大規模疎行列問題をスーパーコンピュータの上で解いています．CG 法を使っているのですが，解きたい問題サイズではメモリがぎりぎり足りません．反復法で，CG 法よりも使用メモリが小さいアルゴリズムを紹介して頂けないでしょうか．行列は実対称です．

1.2　依頼内容の詳細

解きたい問題は，連立一次方程式

$$A\boldsymbol{x} = \boldsymbol{b}, \quad A \in \mathbb{R}^{n\times n}, \quad \boldsymbol{x}, \boldsymbol{b} \in \mathbb{R}^n \tag{1}$$

の解ベクトル $\boldsymbol{x}$ および $\boldsymbol{x}$ と $\boldsymbol{b}$ の内積, $(\boldsymbol{b}, \boldsymbol{x}) := \boldsymbol{b}^{\mathrm{T}}\boldsymbol{x}$ の計算である．行列 A は実対称で,

$$A = (H - \omega I)^2 + cI$$

と書くことができる．行列 H は各非対角要素が $0, 1, -1$ の疎行列であり，現状では行列 H を対称性を考慮せずに非対角要素を以下のように格納している．

- 各行 (i) の非ゼロ要素の列番号 (j) を要素が 1 なら j，要素が-1 なら $-j$ を一次元配列で格納.
- 各行 (i) の先頭の位置を頭出しするために別の配列を格納.

また，対角要素は倍精度の別の一次元配列で格納している．行列サイズは $n = 4 \times 10^9$ 程度で，MPI および OpenMP での並列化を想定している．

現在，Algorithm 1 に示されるように，対角スケーリング前処理付き CG 法を用いて方程式 (1) を求解し，$(\boldsymbol{b}, \boldsymbol{x})$ を計算している．この時，保持する必要のあるベクトルは

$$\boldsymbol{b}, \boldsymbol{x}_k, \boldsymbol{r}_k, \boldsymbol{p}_k, A\boldsymbol{p}_k, H\boldsymbol{p}_k, D$$

の計 7 本である．ただし，$H\boldsymbol{p}_k$ は $A\boldsymbol{p}_k$ の計算過程で temporary として使用する．また，D は行列 A の対角要素で構成される対角行列である．解きたい問題サイズ $n = 4 \times 10^9$ ではメモリの制限で解けないため，省メモリ型の大規模連立方程式解法を紹介して欲しい．

Algorithm 1 共役勾配法 (CG 法) に基づく方法

Set $\boldsymbol{x}_0, \boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0, \boldsymbol{p}_0 = \boldsymbol{r}_0$
Set $D = \mathrm{diag}(A) = \mathrm{diag}((H - \omega I)^2 - cI)$
for $k = 0, 1, 2, \ldots$ **do:**
  $\alpha_k = (\boldsymbol{r}_k, D^{-1}\boldsymbol{r}_k)/(\boldsymbol{p}_k, A\boldsymbol{p}_k)$
  $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
  $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k$
  $\beta_k = (\boldsymbol{r}_{k+1}, D^{-1}\boldsymbol{r}_{k+1})/(\boldsymbol{r}_k, D^{-1}\boldsymbol{r}_k)$
  $\boldsymbol{p}_{k+1} = D^{-1}\boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k$
end for
Compute $(\boldsymbol{b}, \boldsymbol{x}_k)$

2 解決方法

本問題に対する省メモリ化の手法として，以下の 3 つの方法が考えられる．

1. 対称性を考慮した行列の格納方法への変更
2. 内積 $(\boldsymbol{b}, \boldsymbol{x})$ を漸化式で計算する
3. CG 法より使用メモリの少ない解法を用いる

以下に各方法の詳細を示す．

2.1 対称性を考慮した行列の格納方法への変更

行列 H の対称性を考慮し，上三角部分（もしくは下三角部分）のみ保持するようにすることで省メモリ化が可能である．ただし，単純に現在の格納方法で上三角部分のみを格納すると行列ベクトル積 $\boldsymbol{x} = H\boldsymbol{y}$ のノード内並列化の際に，スレッド毎にベクトル $\boldsymbol{x}$ を保持する必要が生じ，使用メモリの増加につながる．

そこで, ブロック化した疎行列格納形式を考える．スレッド数を ns とした時, 行列 H を $ns \times ns$ のブロックに分割し, 各ブロックの $n/ns \times n/ns$ 行列については現在されている疎行列向けの格納形式で保存することで，スレッド毎にベクトルを保持することなく行列ベクトル積の演算を行うことができるようになる．

例えば，行列 H を 3 つの配列 ind, ptr, diag を用いて以下のように格納する．

ind: 狭義上三角部分の各要素のブロック内での列番号（要素が 1 なら j, -1 なら-j）とし，すべてのブロックを通して一次元配列とする．各要素の配置順は図 1 のようにブロック毎に連続となるようにする．配列のサイズは, 狭義上三角部分の非ゼロ要素数である．

ptr: ブロック内での各行の先頭の位置の一次元配列をブロック毎に 3 次元配列として持つ．配列のサイズは $ns \times ns \times (n/ns + 1)$.

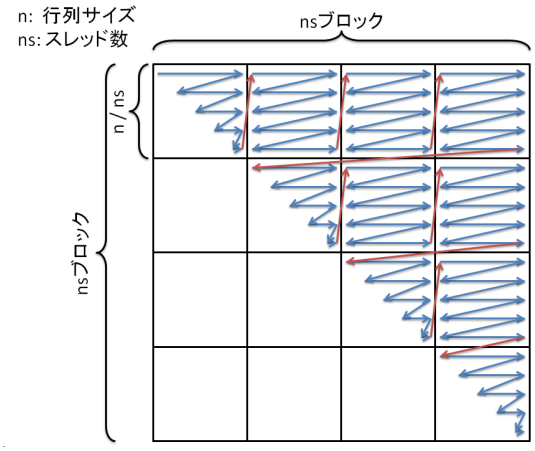

図 1　対称行列向け疎行列格納における ind の順番

diag: 対角要素の一次元配列．サイズは行列の次元 n である．

この時 n が ns で割り切れる場合，行列 H に関する行列ベクトル積 $\boldsymbol{x} = H\boldsymbol{y}$ は Algorithm 2 のように実装される．Algorithm 2 では，

```
do iii = 1, ns
   ...
end do
```

のループが一番大きな部分であるが，変数 iii 毎にベクトル $\boldsymbol{x}$ の参照する要素が重ならないためスレッド毎に $\boldsymbol{x}$ の配列を用意することなく並列化できる．（各 iii に対し，x の (iii-1)*ns から iii*ns の要素のみを参照する．）

上記の方法は省メモリ化の一例であり，疎行列ベクトル積の並列実装に関しては様々な研究が行われている．例えば，[1,2] などを参照されたい．

2.2　内積 $(\boldsymbol{b}, \boldsymbol{x})$ を漸化式で計算する

CG 法のアルゴリズムを基に，内積 $(\boldsymbol{b}, \boldsymbol{x})$ をスカラーの漸化式

$$\eta_{k+1} = \eta_k + \alpha_k(\boldsymbol{r}_k, \boldsymbol{r}_k)$$

で計算する方法が提案されている [3]．この方法では，残差ノルム $\|\boldsymbol{r}_k\|_2$ が十分小さくなった時，$\eta_k \approx (\boldsymbol{b}, \boldsymbol{x})$ となる．文献 [3] の方法に基づき，$\boldsymbol{x}$ および $(\boldsymbol{b}, \boldsymbol{x})$ は Algorithm 3 のように計算される．この方法で保持する必要のあるベクトルは，

$$\boldsymbol{x}_k, \boldsymbol{r}_k, \boldsymbol{p}_k, A\boldsymbol{p}_k, H\boldsymbol{p}_k, D$$

の計 6 本である．もしベクトル $\boldsymbol{x}$ が不要な場合は，$\boldsymbol{x}_k$ の漸化式を省略することで，計算量の削減および更なる省メモリ化が可能である．

Algorithm 2 対称行列向け疎行列ベクトル積

```
% 対角要素に関する計算
do i = 1, n
  x(i) = diag(i) * y(i)
end do

do iii = 1, ns
  % 下三角ブロックに関する計算
  do jjj = 1, iii-1
    do ii = 1, n/ns
      i = (jjj-1)*n/ns + ii
      do jj = ptr(jjj,iii,ii), ptr(jjj,iii,ii+1)-1
        j = (iii-1)*n/ns + abs(ind(jj))
        sign = ind(jj) / abs(ind(jj))
        x(j) = x(j) + sign * y(i)
      end do
    end do
  end do

  % 対角ブロックに関する計算
  jjj = iii
  do ii = 1, n/ns-1
    i = (iii-1)*n/ns + ii
    do jj = ptr(iii,jjj,ii), ptr(iii,jjj,ii+1)-1
      j = (jjj-1)*n/ns + abs(ind(jj))
      sign = ind(jj) / abs(ind(jj))
      x(i) = x(i) + sign * y(j)
      x(j) = x(j) + sign * y(i)
    end do
  end do

  % 上三角ブロックに関する計算
  do jjj = iii+1, ns
    do ii = 1, n/ns
      i = (iii-1)*n/ns + ii
      do jj = ptr(iii,jjj,ii), ptr(iii,jjj,ii+1)-1
        j = (jjj-1)*n/ns + abs(ind(jj))
        sign = ind(jj) / abs(ind(jj))
        x(i) = x(i) + sign * y(j)
      end do
    end do
  end do
end do
```

Algorithm 3 CG 法に基づき内積 $(\boldsymbol{b}, \boldsymbol{x})$ を漸化式で計算する方法

Set $\boldsymbol{x}_0, \boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0, \boldsymbol{p}_0 = \boldsymbol{r}_0, \eta_0 = (\boldsymbol{x}_0, \boldsymbol{b}) + (\boldsymbol{x}_0, \boldsymbol{r}_0)$
Set $D = \mathrm{diag}(A) = \mathrm{diag}((H - \omega I)^2 - cI)$
for $k = 0, 1, 2, \ldots$ **do:**
  $\alpha_k = (\boldsymbol{r}_k, D^{-1}\boldsymbol{r}_k)/(\boldsymbol{p}_k, A\boldsymbol{p}_k)$
  $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
  $\eta_{k+1} = \eta_k + \alpha_k(\boldsymbol{r}_k, D^{-1}\boldsymbol{r}_k)$
  $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k$
  $\beta_k = (\boldsymbol{r}_{k+1}, D^{-1}\boldsymbol{r}_{k+1})/(\boldsymbol{r}_k, D^{-1}\boldsymbol{r}_k)$
  $\boldsymbol{p}_{k+1} = D^{-1}\boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k$
end for

Algorithm 4 最急降下法に基づき内積 $(\boldsymbol{b}, \boldsymbol{x})$ を漸化式で計算する方法

Set $\boldsymbol{x}_0, \boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0, \eta_0 = (\boldsymbol{x}_0, \boldsymbol{b}) + (\boldsymbol{x}_0, \boldsymbol{r}_0)$
Set $D = \mathrm{diag}(A) = \mathrm{diag}((H - \omega I)^2 - cI)$
for $k = 0, 1, 2, \ldots$ **do:**
  $\alpha_k = (\boldsymbol{r}_k, D^{-1}\boldsymbol{r}_k)/(D^{-1}\boldsymbol{r}_k, AD^{-1}\boldsymbol{r}_k)$
  $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k D^{-1}\boldsymbol{r}_k$
  $\eta_{k+1} = \eta_k + \alpha_k(\boldsymbol{r}_k, D^{-1}\boldsymbol{r}_k)$
  $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k AD^{-1}\boldsymbol{r}_k$
end for

Algorithm 5 最小残差法に基づき内積 $(\boldsymbol{b}, \boldsymbol{x})$ を漸化式で計算する方法

Set $\boldsymbol{x}_0, \boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0, \eta_0 = (\boldsymbol{x}_0, \boldsymbol{b}) + (\boldsymbol{x}_0, \boldsymbol{r}_0)$
Set $D = \mathrm{diag}(A) = \mathrm{diag}((H - \omega I)^2 - cI)$
for $k = 0, 1, 2, \ldots$ **do:**
  $\alpha_k = (\boldsymbol{r}_k, AD^{-1}\boldsymbol{r}_k)/(AD^{-1}\boldsymbol{r}_k, AD^{-1}\boldsymbol{r}_k)$
  $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k D^{-1}\boldsymbol{r}_k$
  $\eta_{k+1} = \eta_k + \alpha_k\{2(\boldsymbol{r}_k, D^{-1}\boldsymbol{r}_k) - \alpha_k(D^{-1}\boldsymbol{r}_k, AD^{-1}\boldsymbol{r}_k)\}$
  $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k AD^{-1}\boldsymbol{r}_k$
end for

2.3 CG 法より使用メモリの少ない解法を用いる

CG 法と比べて使用メモリの少ない大規模連立方程式解法として，最急降下法および最小残差法が挙げられる．これらの解法は，ベクトル $\boldsymbol{p}_k$ を用いないためその分のメモリを削減できる．また，各解法に対して文献 [3] のアイディアを導入することで，より省メモリ化されたアルゴリズムを導出できる．最急降下法に基づくアルゴリズムを Algorithm 4 に，また最小残差法に基づくアルゴリズムを Algorithm 5 にそれぞれ示す.

Algorithm 4 および 5 は，どちらも

$$\boldsymbol{x}_k, \boldsymbol{r}_k, A\boldsymbol{r}_k, H\boldsymbol{r}_k, D$$

の 5 本のベクトルのみを必要とするため，Algorithm 1 および 3 より省メモリ化が可能である．ただし，最急降下法および最小残差法は CG 法と比べ，一般に収束性が悪いため演算量が増大することが予想される．

参考文献

[1] T. Katagiri, T. Sakurai, M. Igai, S. Ohshima, H. Kuroda, K. Naono and K. Nakajima, Control formats for unsymmetric and symmetric sparse matrix-vector multiplications on OpenMP implementations, VECPAR 2012, LNCS 7851(2013), 236-248.

[2] M. Krotkiewski and M. Dabrowski, Parallel symmetric sparse matrix-vector product on scalar multi-core CPUs Parallel Computing, 36(2010), 181-198.

[3] Z. Strakoš and P. Tichý, On efficient numerical approximation of the bilinear form $c^*A^{-1}b$, SIAM J. Sci. Comput., 33(2011), 565-587.