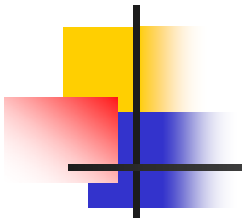


並列アルゴリズム

2005年後期　火曜　2限

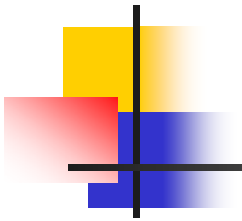
高見　利也（青柳　睦）

Aoyagi@cc.kyushu-u.ac.jp

http://server-500.cc.kyushu-u.ac.jp/

12月13日（火）

9. LU分解法とその並列化（講義）
PCクラスタによる並列プログラミング（演習）



もくじ

LU分解について

- 最も基本的な線形方程式: 未知ベクトルを求める方程式

n元1次連立方程式

$$Ax = b$$

A　n x n 行列(given)

x　nベクトル(unknown)

b　nベクトル(given)

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & & a_{3n} \\ a_{41} & a_{42} & a_{43} & a_{44} & & a_{4n} \\ & \vdots & & & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ \vdots \\ b_n \end{pmatrix}$$

- 数値計算のさまざまな場所で現れる。
- どうやって未知ベクトルを求めるか
  - Gauss消去法(直感的な方法。手計算のやり方)
  - LU分解法(同じ行列に対して繰り返すときに効率的)
  - 繰り返し法(行列にゼロ要素が多い場合など)
  - etc.

LU分解アルゴリズム（基礎知識）

$$A = LU = \begin{pmatrix} l_{11} & & & & & 0 \\ l_{21} & l_{22} & & & \cdots & 0 \\ l_{31} & l_{32} & l_{33} & & & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} & & 0 \\ & \vdots & & & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & l_{n4} & \cdots & l_{nn} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} & & u_{1n} \\ & u_{22} & u_{23} & u_{24} & \cdots & u_{2n} \\ & & u_{33} & u_{34} & & u_{3n} \\ & & & u_{44} & & u_{4n} \\ & \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & u_{nn} \end{pmatrix},$$

$$a_{ij} = \sum_{k=1}^{i} l_{ik} u_{kj} \cdots (i < j)$$

$$a_{ij} = \sum_{k=1}^{j} l_{ik} u_{kj} \cdots (i \geq j)$$

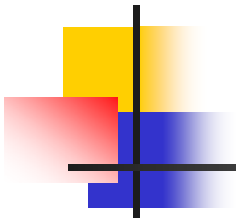
$i < j$ $\quad l_{21}u_{14} + l_{22}u_{24} = a_{24} \quad \Rightarrow u_{24}$

$i = j$ $\quad l_{31}u_{13} + l_{32}u_{23} + l_{33}u_{33} = a_{33} \quad \Rightarrow l_{33}$

$i > j$ $\quad l_{41}u_{12} + l_{42}u_{22} = a_{42} \quad \Rightarrow l_{42}$

$$(i < j) \cdots a_{ij} = \sum_{k=1}^{i-1} l_{ik} u_{kj} + l_{ii} u_{ij} \quad ① \qquad u_{ij} = (a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}) / l_{ii}$$

$$(i \geq j) \cdots a_{ij} = \sum_{k=1}^{j-1} l_{ik} u_{kj} + l_{ij} u_{jj} \quad ② \qquad l_{ij} = (a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}) / u_{jj}$$



LU分解アルゴリズム

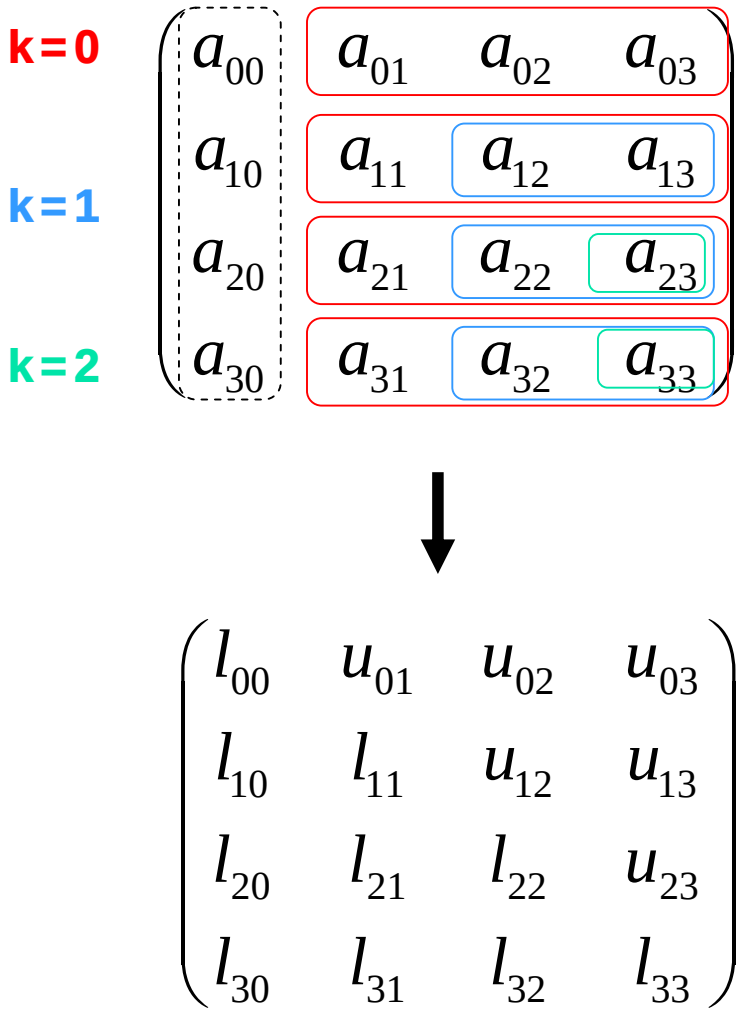
- L,Uの対角値
  - Doolittle型　　$L_{ii}=1$
  - クラウト(Crout)型　　$U_{ii}=1$

- ３通りのアルゴリズム
  - right-looking法（ガウス法）
    メモリアクセスが広範囲，バンド幅を要求
  - left-looking法（ガウス法その２）

  - Crout法
    right-looking法よりもメモリアクセス範囲は狭い

分解アルゴリズムの内容(Right-looking法）

```
/* Right-looking method
Crout type (u_ii == 1)
Input    a: matrix, n: matrix size
Output   a: matrix  */

for(k=0;k<n-1;k++)
 {
  for(j=k+1;j<n;j++)
    a[k][j] /= a[k][k];
  for(i=k+1;i<n;i++)
    for(j=k+1;j<n;j++)
      a[i][j] -= a[i][k] * a[k][j];
 }
```



例えば n=4

```
for(k=0;k<n-1;k++)
  {
   for(j=k+1;j<n;j++)
    a[k][j] /= a[k][k];
   for(i=k+1;i<n;i++)
    for(j=k+1;j<n;j++)
     a[i][j] -= a[i][k] * a[k][j];
  }
```

k=0

$$a_{01} = a_{01} / a_{00} \qquad l\rightarrow a_{21} = a_{21} - a_{20} * a_{01}$$

$$a_{02} = a_{02} / a_{00} \qquad l\rightarrow a_{22} = a_{22} - a_{20} * a_{02}$$

$$a_{03} = a_{03} / a_{00} \qquad u\rightarrow a_{23} = a_{23} - a_{20} * a_{03}$$

$$l\rightarrow a_{11} = a_{11} - a_{10} * a_{01} \qquad l\rightarrow a_{31} = a_{31} - a_{30} * a_{01}$$

$$u\rightarrow a_{12} = a_{12} - a_{10} * a_{02} \qquad l\rightarrow a_{32} = a_{32} - a_{30} * a_{02}$$

$$u\rightarrow a_{13} = a_{13} - a_{10} * a_{03} \qquad l\rightarrow a_{33} = a_{33} - a_{30} * a_{03}$$

k=1

$$a_{12} = a_{12} / a_{11}$$

$$a_{13} = a_{13} / a_{11}$$

$$l\rightarrow a_{22} = a_{22} - a_{21} * a_{12}$$

$$u\rightarrow a_{23} = a_{23} - a_{21} * a_{13}$$

$$l\rightarrow a_{32} = a_{32} - a_{31} * a_{12}$$

$$l\rightarrow a_{33} = a_{33} - a_{31} * a_{13}$$

k=2

$$a_{23} = a_{23} / a_{22}$$

$$a_{33} = a_{33} - a_{32} * a_{23}$$

例えば　n=4

$$(①i<jの場合)u_{ij}=(a_{ij}-\sum_{k=0}^{i-1}l_{ik}u_{kj})/l_{ii}$$

$$(0<1,2,3)u_{01}=(a_{01}-\sum_{k=0}^{0-1}l_{0k}u_{k1})/l_{00}=a_{01}/l_{00}$$

$$u_{02}=a_{02}/l_{00}\qquad u_{03}=a_{03}/l_{00}$$

$$(1<2)u_{12}=(a_{12}-\sum_{k=0}^{1-1}l_{1k}u_{k2})/l_{11}=(a_{12}-l_{10}u_{02})/l_{11}$$

$$(1<3)u_{13}=(a_{13}-\sum_{k=0}^{1-1}l_{1k}u_{k3})/l_{11}=(a_{13}-l_{10}u_{03})/l_{11}$$

（kのループでどの位置にある配列要素が参照されているかに注目）

$$(2<3)u_{23}=(a_{23}-\sum_{k=0}^{2-1}l_{2k}u_{k3})/l_{22}=(a_{23}-l_{20}u_{03}-l_{21}u_{13})/l_{22}$$

$$(②i \geq j\text{の場合}u_{jj}=1\text{に注意})l_{ij}=(a_{ij}-\sum_{k=0}^{j-1}l_{ik}u_{kj})/u_{jj}$$

$$(3,2,1,0\geq 0)l_{00}=(a_{00}-\sum_{k=0}^{0-1}l_{0k}u_{k0})/u_{00}=a_{00}/u_{00}$$

$$l_{10}=a_{10}/u_{00}\quad l_{20}=a_{20}/u_{00}\quad l_{30}=a_{30}/u_{00}$$

$$(3,2,1\geq 1)l_{11}=(a_{11}-\sum_{k=0}^{1-1}l_{1k}u_{k1})/u_{11}=(a_{11}-l_{10}u_{01})/u_{11}$$

$$l_{21}=(a_{21}-l_{20}u_{01})/u_{11}\quad l_{31}=(a_{31}-l_{30}u_{01})/u_{11}$$

$$(3,2\geq 2)l_{22}=(a_{22}-\sum_{k=0}^{2-1}l_{2k}u_{k2})/u_{22}=(a_{22}-l_{20}u_{02}-l_{21}u_{12})/u_{22}$$

$$l_{32}=(a_{32}-l_{30}u_{02}-l_{31}u_{12})/u_{22}$$

$$(3\geq 3)l_{33}=(a_{33}-\sum_{k=0}^{3-1}l_{3k}u_{k3})/u_{33}=(a_{33}-l_{30}u_{03}-l_{31}u_{13}-l_{32}u_{23})/u_{33}$$

アルゴリズムの違い（参考まで）

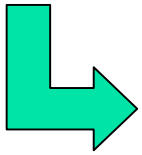
```
/* Left-looking method
(Up-lookuing method)
 Crout type (u_ii == 1)
 Input    a: matrix, n: matrix size
 Output   a: matrix  */

 for(i=0;i<n;i++)
  {
   for(k=0;k<i;k++)
     for(j=k+1;j<n;j++)
       a[i][j] -= a[i][k] * a[k][j];
   for(j=i+1;j<n;j++)
     a[i][j] /= a[i][i];
  }
```

```
/* Crout method
 Crout type (u_ii == 1)
 Input    a: matrix, n: matrix size
 Output   a: matrix  */

 for(k=0;k<n;k++)
  {
   for(i=k;i<n;i++)
     for(j=0;j<k;j++)
       a[i][k] -= a[i][j] * a[j][k];
   for(j=k+1;j<n;j++)
     for(i=0;i<k;i++)
       a[k][j] -= a[i][j] * a[k][i];
   for(j=k+1;j<n;j++)
     a[k][j] /= a[k][k];
  }
```

LU分解の逐次プログラムを並列化する

```
void
lu_s(int n, mat_t a)
{
  int i,j,k;

  for(k=0;k<n-1;k++)
    {
        for(j=k+1;j<n;j++)
            a[k][j] /= a[k][k];

        for(i=k+1;i<n;i++)
          for(j=k+1;j<n;j++)
            a[i][j] -= a[i][k] * a[k][j];
    }
}
```

```
        :
/* Start up MPI */
    MPI_Init(&argc, &argv);

    /* Find out process rank  */
    MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);

/* Find out number of processes */
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
        :
   genmat(n, a );
        :
   lu_p(n, a);
        :
```



並列化

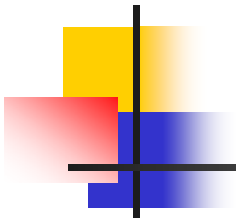
- 行についてcyclic またはblock分割を試みる
- 自Rankが持っていない要素の位置に注意
- 処理が終わったらRank0に集約する

LU分解アルゴリズムの並列化例(R-L法）

```
void  lu_p(int n, mat_t a)
{
 int i,j,k;
 int MAP[SIZE];
 int blk;

 /* cyclic decomposition */
 for(i=0;i<n;i++) MAP[i]= i % numprocs;

 /* block decomposition
 blk=n/numprocs;
 for(i=0;i<numprocs;i++)
   for(j=0;j<blk;j++) MAP[i*blk + j]= i; */

for(k=0;k<n-1;k++)
 {
    if( MAP[k] == my_rank )
       for(j=k+1;j<n;j++)
         a[k][j] /= a[k][k];

 MPI_Bcast(&a[k][k+1],n-k-1,MPI_DOUBLE,
                MAP[k],MPI_COMM_WORLD);
```

（以下右上に続く）

```
    for(i=k+1;i<n;i++)  {
       if( MAP[i] == my_rank )
         for(j=k+1;j<n;j++)
          a[i][j] -= a[i][k] * a[k][j];
    } /* end of i-loop */
 } /* end of k-loop */

 /* copy a[i][*] of my_rank ->  a[i][*] of Rank0  */

    for(i=0;i<n;i++)  {
      if(MAP[i] > 0) MPI_Send(&a[i][0],n,
     MPI_DOUBLE,0,i,MPI_COMM_WORLD);
    }
    if(my_rank==0)
      for(i=0;i<n;i++) {
      if(MAP[i] > 0 ) MPI_Recv(&a[i][0],n,
              MPI_DOUBLE,MAP[i],i,
              MPI_COMM_WORLD,&status);
      }
} /* End of lu_p */
```



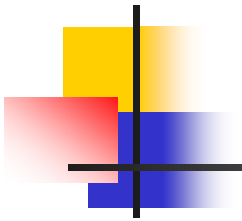
実行時間の計測について

◆ 所要時間（Turn Around Time）

最初に実行を開始したプロセスの開始時刻から、最後に実行を終了したプロセスの終了時刻までを計測し，所要時間とする.

```
MPI_Barrier(MPI_COMM_WORLD);
  t_start = MPI_Wtime();
    /*この間に所要時間を測定する処理を記述*/
  MPI_Barrier(MPI_COMM_WORLD);
  t_stop = MPI_Wtime();

printf("Turn around time = %.16f¥n",
  t_stop  -  t_start  );
```

MPI_Barrier
「バリア同期」と呼ばれ、すべてのプロセスがこれを呼び出すまで各プロセスが待ち合わせる機能を持つ，MPI標準の関数.



PCクラスタを利用した演習（12月13日）

利用計算機：

Xeon 3.06GHz dual processor 16 台から成るPCクラスター

Memory:4GB/node, Disk: 1.2TB(total)

Login UID:

講義中に配布

端末からのLogin 方法：

Windows XP telnet or sshで omega.cc.kyushu-u.ac.jp へ接続

File 転送：Local ⇔ PC Cluster

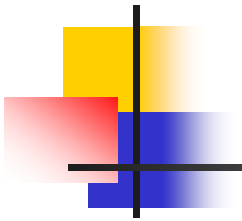
IE(ブラウザ)からftp://UID:PWD@omega.cc.kyushu-u.ac.jp/

File 編集：LocalのWindows上

メモ帳，ワードパッド等，エディター

File 編集：PC Cluster上

vi エディター



PCクラスタを利用した演習(cont.)

Login後の設定:

PCクラスタにはScoreという並列環境が入っており，これをactivateするためにコマンドプロンプトから scout –g all と入力すること.

Mpi program のコンパイル＆リンク:

mpicc　test.c

Mpi program の実行:

mpirun –np [1-16のノード数] a.out　[programの引数]

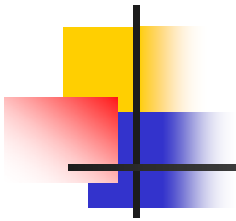
はじめの一歩:

講義で指定したLU分解の並列計算プログラムを翻訳＆実行する

実行時間の計測:

MPI_Barrier(COMM), MPI_Wtime() 関数を使って，時間を計測できるようにprogramを変更し，np 数を変えて経過時間を計る.

MPI_Wtime() 関数の型は　double .



演習レポート(12月13日 出題)

【課題】

講義で使用した行でcyclic分割したLU分解プログラムを基に，
以下について**並列化効率の観点から考察しレポートにして，**
下記の要領でメールで提出してください.

(1) LU分解部分の経過時間を計測できるように書き換え，使用ノード数を変えて計測結果を表またはグラフにまとめる.

(2) 行でBlock分割するようにLU分解プログラムを変更し，上と同様に経過時間を計測し結果を(1)と比較する.

(3) 行でBlock-Cyclic分割するようにLU分解プログラムを変更し，(1)(2)と同様に経過時間を計測し**結果を(1)(2)と比較する**.

提出先：aoyagi@cc.kyushu-u.ac.jp
Subject: 並列アルゴリズム課題
締め切り：平成17年12月27日（火）

形式：text, pdf, ps, Word, Excel ファイルの添付可