CS320/CSE302/ECE392 : INTRODUCTION TO PARALLEL PROGRAMMING FOR SCIENTISTS AND ENGINEERS
Spring 1998

Machine Problem 1

Assigned: Feb. 8, 1999.
Due: Feb. 15, 1999.

Objective:
To explore the vector features of Fortran 90.

In this MP you are requested to translate a Fortran 77 subroutine (listed below) into Fortran 90. The subroutine applies LU decomposition to a matrix.

The LU decomposition of a matrix is useful in finding the inverse of the matrix and in solving a set of linear equations. Given an n*n matrix A, we wish to find two other n*n matrices L and U such that

$$L \cdot U = A$$

or

$$\begin{bmatrix} l_{11} & 0 & 0 & \dots & 0 \\ l_{21} & l_{22} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ l_{n1} & l_{n2} & l_{n3} & \dots & l_{nn} \end{bmatrix} \cdot \begin{bmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & u_{22} & u_{23} & \dots & u_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & u_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix}$$

Here, L is a lower triangular matrix, i.e. $l_{ij} = 0$, i<j and U is an upper triangular matrix, i.e. $u_{ij} = 0$, j<i.

The algorithm given does an in-place computation of the L and U matrices, collapsing the two into a single result matrix as follows

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ l_{21} & u_{22} & u_{23} & \dots & u_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ l_{n1} & l_{n2} & l_{n3} & \dots & u_{nn} \end{bmatrix}$$

The elements l_{ii} are not explicitly stored and are assumed to be 1 for i=1,2,…,n. The elements of this matrix are computed by the subroutine one column at a time from left to right and within each column from top to bottom.

The computation is stopped at the beginning of the subroutine if one of the rows of the matrix is all zeroes.

```
Subroutine LUDCMP(a, n, np, indx, d)
     Given an n*n matrix a, with physical dimension np, this routine replaces it by the LU
decomposition  of a row-wise permutation of itself.  a and n are input, a is output,
arranged as in equation (5); indx is an output vector which records the row permutation
effected by the partial pivoting; d is output as +1 or –1 depending on whether the number
of row interchanges was even or odd, respectively.

Parameter (nmax=100, tiny=1.0E-20)   /*largest expected n, and a small number*/
Dimension a(np, np), indx(n), vv(nmax)  /*vv stores the implicit scaling of each row*./
d=1.0                                                    /*no row interchanges yet*/

Do i=1,n                                                /*loop over rows to get the scaling
information*/
      aamax=0.
      Do j=1,n
            If (abs(a(i,j)) .gt. aamax) aamax=abs(a(i,j))
      Enddo
      If (aamax .eq. 0) then
                      Print *, 'Singular matrix.'  /*no nonzero largest element*/
                      Stop
      Endif
      vv(i)=1.0/aamax                                      /*save the scaling*/
Enddo

Do j=1,n                /*this is the loop over columns of Crout's method*/
      Do i=1, j-1     /*this is equation (3) except for i=j*/
            sum=a(i,j)
            Do k=1,i-1
                  sum=sum-a(i,k)*a(k,j)
             Enddo
             a(i,j)=sum
      Enddo
      aamax=0.     /*initialize for the search for the largest pivot element*/
      Do i=j,n       /*this is i=j for eqn  (3) and i=j+1,…,n for eqn(4)*/
            sum=a(i,j)
            Do k=1,j-1
                  sum=sum-a(i,k)*a(k,j)
            Enddo
            a(i,j)=sum
            dum=vv(i)*abs(sum)         /*figure of merit for the pivot*/
            If (dum .ge. aamax) then   /*is it better than the best so far?*/
                imax=i
                aamax=dum
            Endif
      Enddo
      If (j .ne. imax) then          /*do we need to interchange rows?*/
          Do k=1,n                    /*yes, do so…*/
                dum=a(imax,k)
                a(imax,k)=a(j,k)
                a(j,k)=dum
          Enddo
```

```
          d=-d                          /*and change the parity of d*/
          vv(imax)=vv(j)           /*also, interchange the scale factor*/
      Endif
      indx(j)=imax
      If (a(j,j) .eq. 0) a(j,j)=tiny
      If (j .ne. n) then               /*now divide by the pivot element*/
          dum=1.0/a(j,j)
          Do i=j+1,n
                a(i,j)=a(i,j)*dum
          Enddo
      Endif
 Enddo                                   /*go back for the next column in the reduction*/
 Return
 End subroutine LUDECOMPOSE
```

Rewrite the above subroutine in Fortran90 using as many vector operations as possible.
Some of the functions you might want to keep in mind are:·

`maxval, minval·`
`maxloc, minloc·`
`any, all·`
`spread.`

Also, eliminate the third argument to the subroutine (np) which is not necessary in Fortran 90.

General Instructions:
The machine to be used is newton.cse.uiuc.edu, and the password for your account on this machine has been mailed to you. If you have not received the same, please contact the TA.
The Fortran90 compiler on that machine is located in /opt/SUNWspro/bin. To use the compiler you simply have to type:

f90 <filename.f90>

Make sure that your path variable includes the directory /opt/SUNWspro/bin.
The compiler will produce an a.out file if there are no errors, and you can execute this file by simply typing in its name. Links to the Sun Fortran Manuals are available from the class webpage.
The Fortran 90 compiler in newton does not exploit the parallelism of vector operations. Therefore, your program will be executed sequentially.

Leave your version of the subroutine in your home directory in a file named mp1.f90. It will be picked up from there for grading. Please make sure that your directory is group readable, and DO NOT modify the file after the due date because the date of the file will be taken as the date when you submitted your solution.