TWO-DIMENSIONAL FOURIER TRANSFORMS WITHOUT TRANSPOSING

Robert Clayton

The chief computational cost of F-K migration methods is the calculation of the two-dimensional Fourier transforms. Usually, the data matrix is too large to reside entirely in core memory, thus necessitating the swapping of data between disk and memory. The most direct method of computing the two-dimensional fast Fourier transform for large matrices is:

1) Fourier transform over each of the columns of the matrix by reading a column from the disk, performing a one-dimensional FFT, and writing it back to the disk.

2) Fourier transform over the rows of the matrix by:
   a) transposing the matrix;
   b) one-dimensional FFT over the columns of the transposed matrix as in step 1;
   c) inverse transpose the matrix.

The matrix transpose is the most time-consuming part of this method, and it also has the disadvantage of requiring two extra disk files, each the size of the original matrix. In this report, a simple two-dimensional FFT is presented, which eliminates the transposes from step 2 of the above method. The method allows the Fourier transform to be done "in place," thus eliminating the extra disk files.

The essential idea of the method for the row FFT's is to consider the columns of the matrix to be a series of vectors. A single, one-dimensional FFT is then a special case of the row FFT's, where the vectors are of length one. Thus, a row FFT algorithm can be constructed from a one-dimensional FFT by replacing each scalar (or vector length one) operation with the same operation done over the full length of the column vectors. For example, the inner loop (or twiddle step) of the one-

dimensional FFT subroutine FORK (Claerbout, 1976, p. 12) is:

```
      do 50 i=m, lx, istep
      ctemp= cw*cx(i+l)
      cx(i+l)= cx(i)-ctemp
50    cx(i)= cx(i)+ctempt
```

In the two-dimensional FFT, this step would appear as (in pseudo-Fortran):

```
      do 50 i=m, lx, istep
      call read("read (i)-th column of matrix into vector a")
      call read("read (i+l)-th column of matrix into vector b")
      do 45 k=l, ly
      ctemp= cw*b(k)
      b(k)= a(k)-ctemp
45    a(k)= a(k)+ctemp
      call write("write vector a into (i)-th column of matrix")
50    call write("write vector b into (i+l)-th column of matrix")
```

Here, "read" and "write" are input/output routines to move data in and out of core. If each "scalar" operation in the one-dimensional FFT algorithm is modified in a similar fashion, then the required row FFT algorithm is obtained. To produce a complete two-dimensional FFT, it is also necessary (as in the first method) to do the one-dimensional FFT's over the columns.

At the end of this report a complete adaptation of subroutine FORK for two-dimensional FFT's is given. It is included for illustrative purposes only. The actual program in use at Stanford is coded in a different language.

Further considerations

The row FFT's require $\log_2 N$ passes over the matrix, where N is the number of columns in the matrix. If four or more column vectors can be held in core at one time, then the I/O operations can be considerably reduced by "unfolding" the innermost loop. That is, instead of basing the twiddle step on a Fourier transform of length 2, it could be based on lengths 4, 8, or 16, etc. This would eliminate the I/O operations of the intermediate steps.

The method can be adapted for use with an array processor. The twiddle step which contains all of the floating point operations could be micro-coded for the array processor. This would leave only the I/O operations for the host computer.

I would like to acknowlege Professor Fabio Rocca for the suggestion of this method.

```
      subroutine fork2d(mfile,lx,ly,signx,signy,a,b)
      complex a(lx), b(lx)
c  mfile-       input matrix file number
c  lx-          length of columns (fastest dimension).
c  ly-          length of rows (slowest dimension).
c  signx-       sign of x-tranform.
c  signy-       sign of y-tranform.
c
      complex cw, ctemp, carg, cexp
      sc= sqrt(1.0/ly)
c  do the column FFTs
      do 5 j=1, ly
      call iread(mfile,a,j,lx)
      call fork(a,lx,signx)
      do 4 k=1, lx
4     a(k)= a(k)*sc
5     call iwrite(mfile,a,j,lx)
c
c  do the row FFTs
c
c     do the bit reverse ordering of the data.
      j=1
      do 30 i=1, ly
      if( i.gt.j ) goto 10
      call iread(mfile,a,j,lx)
      call iread(mfile,b,i,lx)
      call iwrite(mfile,b,j,lx)
      call iwrite(mfile,a,i,lx)
10    m= ly/2
20    if( j.le.m) goto 30
      j= j-m
      m= m/2
      if(m.ge.1) goto 20
30    j= j+m
c
      l=1
40    istep= 2*l
      do 50 m=1, l
      carg= (0.,1.)*(3.14159265*signy*(m-1))/l
      cw= cexp(carg)
```

```
c         twiddle step
          do 50 i=m, ly, istep
          call iread(mfile,a,i,lx)
          call iread(mfile,b,i+l,lx)
          do 45 k=1, lx
          ctemp= cw*b(k)
          b(k)= a(k) -ctemp
45        a(k)= a(k) +ctemp
          call iwrite(mfile,a,i,lx)
50        call iwrite(mfile,b,i+l,lx)
          l= istep
          if(l.lt.ly) goto 40
          return
          end
c
          subroutine iread(mfile,vec,irow,lvec)
          complex vec(lvec)
c
c   read in the irow row from matrix.
c
          len= (irow-1)*lvec*8
          iseek= useek(mfile,len,0)
          nread= uread(mfile,vec,lvec*8)
          return
          end
c
          subroutine iwrite(mfile,vec,irow,lvec)
          complex vec(lvec)
c
c   write out the irow row of matrix.
c
          len= (irow-1)*lvec*8
          iseek= useek(mfile,len,0)
          nwrite= uwrite(mfile,vec,lvec*8)
          return
          end
```

REFERENCES

CLAERBOUT, J. F. (1976), *Fundamentals of Geophysical Data Processing* (New York: McGraw-Hill).