12.6 External Storage or Memory-Local FFTs

Sometime in your life, you might have to compute the Fourier transform of a *really large* data set, larger than the size of your computer's physical memory. In such a case, the data will be stored on some external medium, such as magnetic or optical tape or disk. Needed is an algorithm that makes some manageable number of sequential passes through the external data, processing it on the fly and outputting intermediate results to other external media, which can be read on subsequent passes.

In fact, an algorithm of just this description was developed by Singleton[1] very soon after the discovery of the FFT. The algorithm requires four sequential storage devices, each capable of holding half of the input data. The first half of the input data is initially on one device, the second half on another.

Singleton's algorithm is based on the observation that it is possible to bit-reverse 2^M values by the following sequence of operations: On the first pass, values are read alternately from the two input devices, and written to a single output device (until it holds half the data), and then to the other output device. On the second pass, the output devices become input devices, and vice versa. Now, we copy *two* values from the first device, then *two* values from the second, writing them (as before) first to fill one output device, then to fill a second. Subsequent passes read 4, 8, etc., input values at a time. After completion of pass $M - 1$, the data are in bit-reverse order.

Singleton's next observation is that it is possible to alternate the passes of essentially this bit-reversal technique with passes that implement one stage of the Danielson-Lanczos combination formula (12.2.3). The scheme, roughly, is this: One starts as before with half the input data on one device, half on another. In the first pass, one complex value is read from each input device. Two combinations are formed, and one is written to each of two output devices. After this "computing" pass, the devices are rewound, and a "permutation" pass is performed, where groups of values are read from the first input device and alternately written to the first and second output devices; when the first input device is exhausted, the second is similarly processed. This sequence of computing and permutation passes is repeated $M - K - 1$ times, where 2^K is the size of internal buffer available to the program. The second phase of the computation consists of a final K computation passes. What distinguishes the second phase from the first is that, now, the permutations are local enough to do in place during the computation. There are thus no separate permutation passes in the second phase. In all, there are $2M - K - 2$ passes through the data.

Here is an implementation of Singleton's algorithm, based on [1]:

```
      SUBROUTINE fourfs(iunit,nn,ndim,isign)
      INTEGER ndim,nn(ndim),isign,iunit(4),KBF
      PARAMETER (KBF=128)
C     USES fourew
         One- or multi-dimensional Fourier transform of a large data set stored on external media.
         On input, ndim is the number of dimensions, and nn(1:ndim) contains the lengths of
         each dimension (number of complex values), which must be powers of two. iunit(1:4)
         contains the unit numbers of 4 sequential files, each large enough to hold half of the data.
         The four units must be opened for FORTRAN unformatted access. The input data must be
         in FORTRAN normal order, with its first half stored on unit iunit(1), its second half on
         iunit(2), in unformatted form, with KBF real numbers per record. isign should be set
         to 1 for the Fourier transform, to −1 for its inverse. On output, values in the array iunit
         may have been permuted; the first half of the result is stored on iunit(3), the second
         half on iunit(4). N.B.: For ndim > 1, the output is stored by rows, i.e., *not* in FORTRAN
         normal order; in other words, the output is the transpose of that which would have been
         produced by routine fourn.
      INTEGER j,j12,jk,k,kk,n,mm,kc,kd,ks,kr,nr,ns,nv,jx,
     *    mate(4),na,nb,nc,nd
      REAL tempr,tempi,afa(KBF),afb(KBF),afc(KBF)
      DOUBLE PRECISION wr,wi,wpr,wpi,wtemp,theta
      SAVE mate
      DATA mate /2,1,4,3/
```

```
      n=1
      do 11 j=1,ndim
          n=n*nn(j)
          if (nn(j).le.1)
*             pause 'invalid dimension or wrong ndim in fourfs'
      enddo 11
      nv=ndim
      jk=nn(nv)
      mm=n
      ns=n/KBF
      nr=ns/2
      kc=0
      kd=KBF/2
      ks=n
      call fourew(iunit,na,nb,nc,nd)
          The first phase of the transform starts here.
1     continue                          Start of the computing pass.
          theta=3.141592653589793d0/(isign*n/mm)
          wpr=-2.d0*sin(0.5d0*theta)**2
          wpi=sin(theta)
          wr=1.d0
          wi=0.d0
          mm=mm/2
          do 13 j12=1,2
              kr=0
2             continue
                  read (iunit(na)) (afa(jx),jx=1,KBF)
                  read (iunit(nb)) (afb(jx),jx=1,KBF)
                  do 12 j=1,KBF,2
                      tempr=sngl(wr)*afb(j)-sngl(wi)*afb(j+1)
                      tempi=sngl(wi)*afb(j)+sngl(wr)*afb(j+1)
                      afb(j)=afa(j)-tempr
                      afa(j)=afa(j)+tempr
                      afb(j+1)=afa(j+1)-tempi
                      afa(j+1)=afa(j+1)+tempi
                  enddo 12
                  kc=kc+kd
                  if (kc.eq.mm) then
                      kc=0
                      wtemp=wr
                      wr=wr*wpr-wi*wpi+wr
                      wi=wi*wpr+wtemp*wpi+wi
                  endif
                  write (iunit(nc)) (afa(jx),jx=1,KBF)
                  write (iunit(nd)) (afb(jx),jx=1,KBF)
              kr=kr+1
              if (kr.lt.nr) goto 2
              if(j12.eq.1.and.ks.ne.n.and.ks.eq.KBF) then
                  na=mate(na)
                  nb=na
              endif
              if (nr.eq.0) goto 3
          enddo 13
3     call fourew(iunit,na,nb,nc,nd)           Start of the permutation pass.
      jk=jk/2
4     if (jk.eq.1) then
          mm=n
          nv=nv-1
          jk=nn(nv)
      goto 4
      endif
      ks=ks/2
      if (ks.gt.KBF) then
          do 16 j12=1,2
```

```
            do 15 kr=1,ns,ks/KBF
                do 14 k=1,ks,KBF
                    read (iunit(na)) (afa(jx),jx=1,KBF)
                    write (iunit(nc)) (afa(jx),jx=1,KBF)
                enddo 14
                nc=mate(nc)
            enddo 15
            na=mate(na)
        enddo 16
        call fourew(iunit,na,nb,nc,nd)
        goto 1
    else if (ks.eq.KBF) then
            nb=na
            goto 1
    endif
continue
j=1
```
The second phase of the transform starts here. Now, the remaining permutations are sufficiently local to be done in place.
```
5   continue
        theta=3.141592653589793d0/(isign*n/mm)
        wpr=-2.d0*sin(0.5d0*theta)**2
        wpi=sin(theta)
        wr=1.d0
        wi=0.d0
        mm=mm/2
        ks=kd
        kd=kd/2
        do 18 j12=1,2
            do 17 kr=1,ns
                read (iunit(na)) (afc(jx),jx=1,KBF)
                kk=1
                k=ks+1
6               continue
                    tempr=sngl(wr)*afc(kk+ks)-sngl(wi)*afc(kk+ks+1)
                    tempi=sngl(wi)*afc(kk+ks)+sngl(wr)*afc(kk+ks+1)
                    afa(j)=afc(kk)+tempr
                    afb(j)=afc(kk)-tempr
                    afa(j+1)=afc(kk+1)+tempi
                    afb(j+1)=afc(kk+1)-tempi
                    j=j+2
                    kk=kk+2
                if (kk.lt.k) goto 6
                kc=kc+kd
                if (kc.eq.mm) then
                    kc=0
                    wtemp=wr
                    wr=wr*wpr-wi*wpi+wr
                    wi=wi*wpr+wtemp*wpi+wi
                endif
                kk=kk+ks
                if (kk.le.KBF) then
                    k=kk+ks
                    goto 6
                endif
                if (j.gt.KBF) then
                    write (iunit(nc)) (afa(jx),jx=1,KBF)
                    write (iunit(nd)) (afb(jx),jx=1,KBF)
                    j=1
                endif
            enddo 17
            na=mate(na)
        enddo 18
        call fourew(iunit,na,nb,nc,nd)
```

```
      jk=jk/2
   if (jk.gt.1) goto 5
   mm=n
7  if (nv.gt.1) then
      nv=nv-1
      jk=nn(nv)
      if (jk.eq.1) goto 7
      goto 5
   endif
   return
   END
```

```
   SUBROUTINE fourew(iunit,na,nb,nc,nd)
   INTEGER na,nb,nc,nd,iunit(4),ii
      Utility used by fourfs. Rewinds and renumbers the four files.
   do 11 ii=1,4
      rewind(unit=iunit(ii))
   enddo 11
   ii=iunit(2)
   iunit(2)=iunit(4)
   iunit(4)=ii
   ii=iunit(1)
   iunit(1)=iunit(3)
   iunit(3)=ii
   na=3
   nb=4
   nc=1
   nd=2
   return
   END
```

For one-dimensional data, Singleton's algorithm produces output in exactly the same order as a standard FFT (e.g., four1). For multidimensional data, the output is the *transpose* of the conventional arrangement (e.g., the output of fourn). This peculiarity, which is intrinsic to the method, is generally only a minor inconvenience. For convolutions, one simply computes the component-by-component product of two transforms in their nonstandard arrangement, and then does an inverse transform on the result. Note that, if the lengths of the different dimensions are not all the same, then you must reverse the order of the values in nn(1:ndim) (thus giving the transpose dimensions) before performing the inverse transform. Note also that, just like fourn, performing a transform and then an inverse results in multiplying the original data by the product of the lengths of all dimensions.

We leave it as an exercise for the reader to figure out how to reorder fourfs's output into normal order, taking additional passes through the externally stored data. We doubt that such reordering is ever really needed.

You will likely want to modify fourfs to fit your particular application. For example, as written, KBF $\equiv 2^K$ plays the dual role of being the size of the internal buffers, and the record size of the unformatted reads and writes. The latter role limits its size to that allowed by your machine's I/O facility. It is a simple matter to perform multiple reads for a much larger KBF, thus reducing the number of passes by a few.

Another modification of fourfs would be for the case where your virtual memory machine has sufficient address space, but not sufficient physical memory, to do an efficient FFT by the conventional algorithm (whose memory references are extremely nonlocal). In that case, you will need to replace the reads, writes, and rewinds by mappings of the arrays afa, afb, and afc into your address space. In other words, these arrays are replaced by references to a single data array, with offsets that get modified wherever fourfs performs an I/O operation. The resulting algorithm will have its memory references local within blocks of size KBF. Execution speed is thereby sometimes increased enormously, albeit at the cost of requiring twice as much virtual memory as an in-place FFT.

CITED REFERENCES AND FURTHER READING:

Singleton, R.C. 1967, *IEEE Transactions on Audio and Electroacoustics*, vol. AU-15, pp. 91–97. [1]

Oppenheim, A.V., and Schafer, R.W. 1989, *Discrete-Time Signal Processing* (Englewood Cliffs, NJ: Prentice-Hall), Chapter 9.