A COMPLEX TRIDIAGONAL MATRIX SOLVER

Jeff Thorson

The heart of many of our wave-equation migration schemes is the solution of a tridiagonal system of equations. Since this routine may lie inside two or more nested program loops, it would be worthwhile to utilize the speed of an array processor in solving it.

We now have a Floating Point Systems AP-120B array processor with 512 words of standard program memory and 8,000 words of data memory. A subroutine to solve the most general tridiagonal system -- with complex elements -- has been written in AP assembly language and is included in this report.

Together with a vector function chaining option which allows several routines to be linked together inside the array processor, it is possible to transfer a large portion of a migration program from the host to the AP, despite the small size of our program memory.

The subroutine works on the Floating Point Systems standard memory option; it has not been tested on AP-120B fast memory. All relevant documentation is in the program listing.

```
"************ CTRI = COMPLEX TRIDIAGONAL MATRIX SOLVE ****************
        $title ctri
        $entry ctri,5
        $ext div
"
"Abstract:
"       This subroutine solves a general complex tridiagonal system
"       of the following form:
"
"       [ b(0) a(0)                          ] [ t(0) ]   [ d(0) ]
"       [ c(1) b(1) a(1)                     ] [ t(1) ]   [ d(1) ]
"       [      c(2) b(2) a(2)                ] [ t(2) ]   [ d(2) ]
"       [           c(3)   .   .             ] [   .  ] = [   .  ]
"       [                  .   .   a(n-2)    ] [   .  ]   [   .  ]
"       [                c(n-1) b(n-1)       ] [t(n-1)]   [d(n-1)]
"
"Statistics:
"       AP-120B with standard memory
"       PS - 56 instructions
"       Speed - 10.1 us/complex point
"       Author - J.R. Thorson  July 1978
"
"Subroutines used:
"       div
"
"Scratch:
"       sp(0-7,13-15)
"       dpx(-4 to 3)
"       dpy(-4 to 3)
"       dpa unchanged
"       tm - used
"       md - answer is overwritten onto input vectors:
"               e(i) onto c(i)
"               f(i) onto b(i+1)
"               t(i) onto d(i)
"               c(0) and a(n-1) are not used.
"
"Useage:
"       call ctri(a,b,c,d,n)
"       (or) jsr ctri                       S-PAD
"       a = base address of vector a          0
"       b = base address of vector b          1
"       c = base address of vector c          2
"       d = base address of vector d          3
"       n = size of system                    4
"
"Algorithm:
"----------
"       a,b,c,d,e,f,g,h,t are complex.
"       g = b(0)
"       h = d(0)
"       do forward loop,  k = 0,n-1
"               x = g(real)**2 + g(imag)**2
"               x = 1/x  (jsr div)
"               g = x * g
"               e(k) = g * a(k)
```

```
"                f(k) = g * h
"                g = b(k+1) - e(k)*c(k+1)
"                h = d(k+1) - f(k)*c(k+1)
"                continue
"        t(n-1) = f(n-1)
"        do reverse loop, k = n-2,1
"                t(k) = f(k) - e(k)*t(k+1)
"                continue
"        end
"
"Notes:
"        See Claerbout, FGDP, pp. 188-189 for derivation.
"        Here e(k) is opposite in sign with the e(k) in Claerbout.
"        Treatment of boundary conditions:
"                c(0) and a(n-1) are not used in the algorithm and
"                can be any value.  Adjust a(0),b(0),d(0),c(n-1),
"                b(n-1) and d(n-1) to set the desired b.c. --
"                1) e(0)=a(0)/b(0)
"                2) f(0)=d(0)/b(0)
"                3) t(n-1)=(d(n-1)-f(n-2)*c(n-1))/(b(n-1)-e(n-2)*c(n-1))
"        For many applications vector c equals vector a.  c may overlay
"        a in the AP memory -- i.e.,
"                call apput(c,aptr,n,2)
"                ...
"                call ctri(aptr,bptr,aptr,dptr,n)        will work.
"
"
"        S-Pad:
"        ------
         aptr    =   0
         bptr    =   1
         cptr    =   2
         dptr    =   3
         n       =   4
         eptr    =   2
         fptr    =   1
         tptr    =   3
         count   =   5
         xinc    =   6
"
"        Data pad mnemonics:
"
"        DPX
"        ---
         hr   =  -4                      "real part of h
         hi   =  -3                      "imag part of h
         ar   =  -2                      "etc.
         ai   =  -1
         tr   =   0                      "tr,ti are used in reverse loop,
         ti   =   1                      "otherwise DPX 0,1 are scratch
         cr   =   2
         ci   =   3
"
"        DPY
"        ---
         gr   =  -4
```

```
        gi   =  -3
        er   =  -2
        ei   =  -1
        fr   =   2
        fi   =   3
"
ctri:   mov bptr,bptr; setma            "fetch b(0)r
        ldspi 7; db=!one                "store address of 1.0 in sp 7
        mov n,count;                    "set counter to n
          incma                         "fetch b(0)i
        mov 7,7; settma;                "fetch 1.0 out of table memory
          dpy(gr)<md                    "b(0)r is ready, gr = b(0)r
        mov dptr,dptr; setma;           "fetch d(0)r
          fmul dpy(gr),dpy(gr)          "form gr**2
        clr xinc;                       "to test n
          dpy(gi)<md;                   "b(0)i is ready, gi = b(0)i
          fmul                          "push.
        sub n,xinc;                     "negate n - see next instruction
          fmul dpy(gi),dpy(gi);         "form gi**2
          dpy(0)<tm;                    "place 1.0 into dpy
          incma                         "fetch d(0)i
        bge error;                      "branch to error if n <= 0
          fadd fm,zero;                 "gr**2 goes into the adder
          dpx(hr)<md;                   "d(0)r is ready, hr = d(0)r
          fmul                          "push.
        mov aptr,aptr; setma;           "fetch a(0)r
          fadd;                         "push out gr**2
          fmul                          "push out gi**2
        ldspi xinc; db=2;               "xinc equals 2
          fadd fm,fa                    "form gr**2 + gi**2
        fadd;                           "push.
          dpx(hi)<md                    "d(0)i is ready, hi = d(0)i
        dpx(ar)<md;                     "a(0)r is ready, ar = a(0)r
          br in                         "branch into forward loop
"
error:  clr n; return                   "possible errors:
                                        "1.   n <= 0
                                        "2.   divide by zero in div
                                        "3.   floating point overflow.
                                        "n is reset to zero in each case.
"
"       The preliminaries are done:  gr,gi,hr,hi,ar have been placed
"       in the data pad, gr**2 + gi**2 rests in the adder output  buffer,
"       and 1.0 is in dpy(0) for the use of subroutine div.
"       Pseudo-counter k = 0 (used in the comments).
"
forward: mi<dpy(ei); incma;             "write e(k-1)i into memory.
          fadd                          "push.  This program step has
                                        "been moved from the bottom of
                                        "the loop to allow branch beq
                                        "below to reach to 'neutral'.
in:     dpx(0)<fa;                      "position gr**2 + gi**2,
          jsr div                       "and divide.
        inc# aptr; setma;               "fetch a(k)i
          fmul dpx(0),dpy(gr)           "gr = division result * gr
        bfpe error;                     "goto error if divisor was zero
```

```
  mov 7,7; settma;          "fetch 1.0 again from table memory
  fmul dpx(0),dpy(gi)       "gi = division result * gi
mov bptr,bptr;              "set spfn to point to b(k)r,
  mi<dpy(fr); setma;        "write f(k-1)r onto b(k)r
  fmul                      "push.
fmul fm,dpx(ar);            "gr * ar
  dpx(ai)<md;               "a(k)i is ready, ai = a(k)i
  dpy(gr)<fm                "store gr in data pad y
mi<dpy(fi); incma;          "write f(k-1)i into b(k)i
  fmul fm,dpx(ar);          "gi * ar
  dpy(gi)<fm                "store gi in data pad y
fmul dpx(ai),dpy(gi)        "ai * gi
add xinc,cptr; setma;       "fetch c(k+1)r
  fadd fm,zero;             "ar*gr into adder
  fmul dpx(ai),dpy(gr)      "ai * gr
fsubr fm,zero;              "-ar*gi into adder
  fmul dpx(hr),dpy(gr)      "hr * gr
dec count;                  "decrement counter
  fadd fm,fa;               "ai*gi + ar*gr
  fmul dpx(hi),dpy(gr);     "hi * gr
  incma                     "fetch c(k+1)i
beq neutral;                "branch out if count = 0
  fadd fm,fa;               "ai*gr - ar*gi
  fmul dpx(hi),dpy(gi);     "hi * gi
  dpx(cr)<md                "c(k+1)r is ready, cr = c(k+1)r
add xinc,bptr; setma;       "fetch b(k+1)r
  fadd fm,zero;             "hr*gr into adder
  fmul dpx(hr),dpy(gi);     "hr * gi
  dpy(er)<fa                "e(k)r = ar*gr + ai*gi
fadd fm,zero;               "hi*gr into adder
  fmul dpx(cr),dpy(er);     "cr * er
  dpx(ci)<md;               "ci = c(k+1)i
  dpy(ei)<fa                "e(k)i = ai*gr - ar*gi
fadd fm,fa;                 "hi*gi + hr*gr
  fmul dpx(cr),dpy(ei);     "cr * ei
  incma                     "fetch b(k+1)i
fsubr fm,fa;                "-hr*gi + hi*gr
  fmul dpx(ci),dpy(ei)      "ci * ei
add xinc,dptr; setma;       "fetch d(k+1)r
  fsubr fm,md;              "-cr*er + b(k+1)r
  fmul dpx(ci),dpy(er);     "ci * er
  dpy(fr)<fa                "store f(k)r in dpy
fsubr fm,md;                "-cr*ei + b(k+1)i
  fmul dpx(cr),dpy(fr);     "cr * fr
  dpy(fi)<fa                "store f(k)i in dpy
fadd fm,fa;                 "ci*ei - cr*er + br
  fmul dpx(cr),dpy(fi);     "cr * fi
  incma                     "fetch d(k+1)i
fsubr fm,fa;                "-ci*er - cr*ei + bi
  fmul dpx(ci),dpy(fi)      "ci * fi
add xinc,aptr; setma;       "fetch a(k+1)r
  fsubr fm,md;              "-cr*fr + d(k+1)r
  fmul dpx(ci),dpy(fr);     "ci * fr
  dpy(gr)<fa                "store new gr in dpy
fsubr fm,md;                "-cr*fi + d(k+1)i
  fmul dpy(gr),dpy(gr);     "square gr
```

```
          dpy(gi)<fa                        "store new gi in dpy
        fadd fm,fa;                         "ci*fi - cr*fr + dr
          fmul dpy(gi),dpy(gi);             "square gi
          dpy(0)<tm                         "reload 1.0 into dpy - for div
        fsubr fm,fa;                        "-ci*fr - cr*fi + di
          fmul;                             "push,
          dpx(ar)<md                        "a(k+1)r is ready, ar = a(k+1)r
        sub# xinc,cptr;                     "set spfn to c(k)r,
          mi<dpy(er); setma;                "and write e(k)r over c(k)r
          fadd;                             "push adder,
          fmul;                             "push multiplier,
          dpx(hr)<fa;                       "store new hr in dpx
          dpy(1)<fm                         "temporary place for gr**2
        fadd fm,dpy(1);                     "gi**2 + gr**2
          dpx(hi)<fa;                       "store new hi in dpx
          jmp forward                       "loop back. Also k = k+1
"
"       Status:
"       n writes of fi,fr have been made to memory:
"       (   0,0    f(0)r,i   ...   f(n-2)r,i )  overlay
"       ( b(0)r,i  b(1)r,i   ...   b(n-1)r,i )
"       n-1 writes of ei,er have been made:
"       ( e(0)r,i  e(1)r,i   ...   e(n-2)r,i )  overlay
"       ( c(0)r,i  c(1)r,i   ...   c(n-2)r,i )
"       Note f(n-2)r,i and e(n-2)r,i are still in DPY.
"       f(n-1)r,i are in the pipe, = t(n-1)r,i.
"       fptr = bptr points to b(n-1)r = f(n-2)r
"       eptr = cptr   '    '  c( n )r = e( n )r
"       tptr = dptr   '    '  d(n-1)r
"
neutral: sub xinc,eptr                      "eptr points to c(n-1)r now.
        add xinc,tptr;                      "reposition tptr to d(n)r
          fadd fm,zero;                     "hr*gr into adder
          fmul dpx(hr),dpy(gi)              "hr * gi
        sub xinc,eptr;                      "reposition eptr to c(n-2)r
          fadd fm,zero;                     "hi*gr into adder
          fmul                              "push.
        inc# fptr; setma;                   "fetch f(n-2)i - fptr is ok
          fadd fm,fa;                       "f(n-1)r = hi*gi + hr*gr
          fmul                              "push.
        mov n,count;                        "set counter to n
          fsubr fm,fa                       "f(n-1)i = -hr*gi + hi*gr
"
"       Reverse loop:  pseudo-counter k = n-1
"
reverse: sub xinc,tptr;                     "decrement tptr,
          mi<fa; setma;                     "write t(k)r from adder
          dpx(tr)<fa;                       "also store t(k)r in DPX
          fadd                              "push out t(k)i
        dpy(fi)<md                          "f(k-1)i is ready, store in DPY
        mi<fa; incma;                       "write t(k)i
          dpx(ti)<fa                        "also store t(k)i in DPX
        fmul dpx(tr),dpy(er)                "tr * er
        sub xinc,eptr; setma;               "fetch e(k-2)r
          fmul dpx(ti),dpy(er)              "ti * er
        dec count;                          "decrement counter - test is next
```

```
          fmul dpx(ti),dpy(ei)          "ti * ei
        beq done;                       "branch out if counter = 0
          fsubr fm,zero;                "-er*tr into adder
          fmul dpx(tr),dpy(ei);         "tr * ei
          incma                         "fetch e(k-2)i
        fsubr fm,zero;                  "-er*ti into adder
          fmul;                         "push,
          dpy(er)<md                    "e(k-2)r is ready, store in DPY
        sub xinc,fptr; setma;           "fetch f(k-2)r
          fadd fm,fa;                   "-er*tr + ei*ti
          fmul                          "final push.
        fsubr fm,fa;                    "-er*ti - ei*tr
          dpy(ei)<md                    "e(k-2)i is ready, store in DPY
        fadd dpy(fr),fa;                "t(k-1)r = fr - er*tr + ei*ti
          incma                         "fetch f(k-2)i
        fadd dpy(fi),fa;                "t(k-1)i = fi - er*ti - ei*tr
          dpy(fr)<md;                   "f(k-2)r is ready, store in DPY
          br reverse                    "loop back - and k = k-1
"
done:   return
        $end
```