# So you went and bought a vector computer

*Stewart A. Levin*

## FROM THE MAILBOX

From: rick
Subject: Latest benchmark results convex vs vax

| | | | |
|---|---|---|---|
| Fork (textbook fft) | 5.2x | Rick | |
| Fft | 7.0x | Bill | |
| Fft Singleton | 62.5 | Rick | |
| stew's fft (F) | 3.2x | Li | most time in I/O |
| diskfft (F) | 5.5x | Li | |
| jon's fft (F) | 5x | Rick | most loops couldn't vectorize |

From: convexw!jensen (Bill Jensen)
Subject: FFT

The message is that complex arithmetic CAN be slower than doing it with reals on a convex. There are three reasons why I wrote this FFT the way it is. The first reason is that the original Cooley-Tukey algorithm that I was replacing was written this way. By replicating the algorithm, I was able to check out the two FFTs side by side in order to make sure mine was working properly.

The second reason has to do with performance. The C-1 is faster when it is working with contiguous arrays. I surmised that by splitting the real and the imaginary parts like I did, I would get better performance. I do not know if this is the case, because I have not tested it. I should know by the end of the day, though, because I am going to rewrite the assembler FFT to follow the FORTRAN algorithm. This is something the compiler cannot do, because it will always have to assume that complex numbers will be exported to other routines.

The third reason is that the optimization in pass 2 to eliminate multiplies would not be possible with complex arithmetic. Multiplying by (0.0, -1.0)

would result in four multiplies and two additions. Someday, the compiler might be able to optimize this.

From: convexw!jensen (Bill Jensen)
Subject: General FFT

I have updated fft512.f to generalize it for you. It is untested at this time, but I could check it out after Monday. It can be found in /mnt/stanford/generalfft.f . As written, it will do 8 point FFTs to 1024 point FFTs. Larger FFTs are easily accomplished just by changing the parameter statement.

I am sorry I do not have time to check this out now. I am busy getting the 512 pt assembler FFT to work.

P.S. I also removed the "funny" array declarations. However, I do believe that declaring twi(256,3:9) is a legal F77 construct. I have seen this in other compilers (Apollo and Data General).

From: Richard Ottolini <rick>
Subject: Interesting fft timing results:

These results are preliminary and shouldn't be sent to Convex.
They need to be double checked for both speed and accuracy.

1024 pt complex ffts: (averaged over 1000)
  6.5 ms/fft  Convex (jensen; fc -O2)
 25 ms/fft  FPS (vector chained in groups of 32, including put/gets)
400 ms/fftVAX (jensen; f77 -O)
400 ms/fftVAX (seplib: clayton; cc)
880 ms/fftVAX (fork; f77 -O)

From: convexw!jensen (Bill Jensen)
Subject: FFT

In response to the issues raised in the READ_ME file:

1) "confft.f" and "coeff.f" now work. I fixed several typos and errors that were made in typing it in and converting it to general form. In addition, I changed the logic in "coeff.f" that was rewritten to eliminate possible nonportable logical

operations.

2) The routines should work for any length of FFT from 8 points to 1024 points. This feature is not thoroughly tested. At this time the maximum size FFT is set to 1024 and is controlled by the parameter line IN BOTH "confft.t" and "coeff.f". These parameters must be set properly for the FFT routines to work properly. "len" must be the largest fft that is to be done. "len2" must be "len" divided by two. "lenlg2" is log to the base two of "len", i.e. "2**lenlg2 .eq. len".

3) The routines now provide the correct answer with one notable difference! The output of the convex FFT is scaled by 32. from the output of the "fork" FFT. I looked at your code and noticed that the bit reversal logic scales all of the input values by "sqrt (1./lw)" which is 32. (lw = 1024). I do not know why this is there, since this is not part of a standard fft routine. I did modify the code in "confft.f" to scale all of the inputs, and, once that was done, the answers seemed to agree.

From: stew
Subject: compiler optimizations and vectorized FFT's

I just read an article in the new journal "Parallel Computing" published by North-Holland entitled "FFT algorithms for vector computers". This article is clear and readable and contains some illustrative FORTRAN codes. (V.1, 1984, p45-63, Paul N. Swartztrauber). I typed them into the computer and compiled them on the convex in directory /mnt/stanford/ffts. I've inserted the compiler's vectorization comments in these sources and generated the assembly code files also. Perhaps the most striking comments are in stock.vect.f in which loop inversion is explicitly used to insure that the inner loop is of length at least sqrt(N). The compiler fully vectorized the first, uninverted loop by inverting it and distributing it (also thereby defeating the intent of explicit loop inversion). However the following, manually inverted, loops were not fully vectorized! Looks like more work is needed on the optimization logic.

## DISCUSSION

Swarztrauber, in the abovementioned article, lists six general factors which influence performance of a vector machine and discusses several FFT algorithms in light of them. These factors are

(a) Pipeline start-up time

(b) Recurrences and computational dependencies

(c) Instruction ordering and scheduling

(d) Efficient data arrangement and access in memory

(e) Vectorization by compilers

(f) Selection of algorithm matched to architecture

He shows that classical FFT algorithms, mostly variations on the textbook, Cooley-Tukey textbook algorithm (e.g. Appendix A or routine FORK in FGDP p.12), run inefficiently on most vector computers for reasons (a) and (d): many short loops accessing noncontiguous data. These factors are not critical on the FPS because it contains special purpose hardware for bit-reverse addressing and sine/cosine table lookup. Swarztrauber then presents sample codes for some alternative FFT algorithms. I heartily recommend you read his article. In Appendix A I've included listings of several Fortran subroutines copied (with corrections) from or based upon Swarztrauber's paper. I've also included timings for a 1024 point complex input vector.

## REFERENCES

Levin, S.A., 1983, Matrix transposition and the 2-D FFT revisited: SEP-35, p.249-260

Rose, D.J., 1980, Matrix identities of the fast Fourier transform: Linear algebra and its applications, 29 p.423-443

Swarztrauber, P.N., 1984, FFT algorithms for vector computers: Parallel Computing, 1 p.45-63

## APPENDIX A

```
      SUBROUTINE CTSORT(M,C,WORK)
C
C     ORDER PHASE FOR THE COOLEY-TUKEY FFT
C
      COMPLEX C(1),WORK(1)
      N=2**M
      DO 100 L=1,M
      NS=2**(L-1)
      LS=N/(NS+NS)
      CALL CTSRT1(LS,NS,C,WORK)
      DO 100 I=1,N
      C(I)=WORK(I)
100   CONTINUE
      RETURN
      END
      SUBROUTINE CTSRT1(LS,NS,C,CH)
      COMPLEX C(2,LS,NS),CH(LS,2,NS)
      DO 200 J=1,NS
      DO 200 I=1,LS
      CH(I,1,J)=C(1,I,J)
      CH(I,2,J)=C(2,I,J)
200   CONTINUE
      RETURN
      END
```

FIG. 1. Cooley-Tukey sort phase. 25 msec for 1024 complex input. This is more general but not as efficient as bit-reversal addressing for power of two FFT's. In this subroutine the bits are reversed one at a time in log2 N shuffles of the data.

```
      SUBROUTINE CTCMBN(IS,M,C)
C
C     COMBINE PHASE FOR THE COOLEY-TUKEY FFT
C
      COMPLEX C(1)
      N=2**M
      DO 100 L=1,M
      LS=2**(L-1)
      NS=N/(LS+LS)
      CALL CTCMB1(IS,LS,NS,C)
100   CONTINUE
      RETURN
      END
      SUBROUTINE CTCMB1(IS,LS,NS,C)
      COMPLEX OMEGA,OMEGK,WYK,C(LS,2,NS)
      ANGLE=FLOAT(IS)*4.*ATAN(1.)/FLOAT(LS)
      OMEGA=CMPLX(COS(ANGLE),SIN(ANGLE))
      OMEGK=1.
      DO 300 I=1,LS
      DO 200 J=1,NS
      WYK=OMEGK*C(I,2,J)
      C(I,1,J)=C(I,1,J)+WYK
      C(I,2,J)=C(I,1,J)-WYK
200   CONTINUE
      OMEGK=OMEGA*OMEGK
300   CONTINUE
      RETURN
      END
```

FIG. 2. Cooley-Tukey combine phase. 40 msec for 1024 complex input. Computations are done in place. Starts with bit-reverse ordered data, produces serially ordered output.

```
      SUBROUTINE GSCMBN(IS,M,C)
C
C     GENTLEMAN-SANDE FFT COMBINE PHASE
C
      COMPLEX C(1)
      N=2**M
      DO 100 L=1,M
      NS=2**(L-1)
      LS=N/(NS+NS)
      CALL GSCMB1(IS,LS,NS,C)
100   CONTINUE
      RETURN
      END
      SUBROUTINE GSCMB1(IS,LS,NS,C)
      COMPLEX OMEGA,OMEGK,WYK,C(LS,2,NS)
      ANGLE=FLOAT(IS)*4.*ATAN(1.)/FLOAT(LS)
      OMEGA=CMPLX(COS(ANGLE),SIN(ANGLE))
      OMEGK=1.
      DO 300 I=1,LS
      DO 200 J=1,NS
      WYK=C(I,1,J)-C(I,2,J)
      C(I,1,J)=C(I,1,J)+C(I,2,J)
      C(I,2,J)=OMEGK*WYK
200   CONTINUE
      OMEGK=OMEGA*OMEGK
300   CONTINUE
      RETURN
      END
```

FIG. 3. Gentleman-Sande combine phase. 40 msec for 1024 complex input. Starts with serially ordered input, yields bit-reverse ordered output.

```
      SUBROUTINE GSSORT(M,C,WORK)
C
C     ORDER PHASE FOR THE GENTLEMAN-SANDE FFT
C
      COMPLEX C(1),WORK(1)
      N=2**M
      DO 100 L=1,M
      LS=2**(L-1)
      NS=N/(LS+LS)
      CALL GSSRT1(LS,NS,C,WORK)
      DO 100 I=1,N
      C(I)=WORK(I)
100   CONTINUE
      RETURN
      END
      SUBROUTINE GSSRT1(LS,NS,C,CH)
      COMPLEX C(LS,2,NS),CH(2,LS,NS)
      DO 200 J=1,NS
      DO 200 I=1,LS
      CH(1,I,J)=C(I,1,J)
      CH(2,I,J)=C(I,2,J)
200   CONTINUE
      RETURN
      END
```

FIG. 4. Gentleman-Sande sort phase. 24 msec for 1024 complex input. Input is bit-reverse ordered, output is serially ordered.

```
      SUBROUTINE PSORT(M,C,WORK)
C
C     ORDER PHASE FOR THE PEASE FFT
C
      COMPLEX C(1),WORK(1)
      N=2**M
      DO 100 L=1,M
      NS=2**(L-1)
      LS=N/(NS+NS)
      CALL PSRT1(LS,NS,C,WORK)
      DO 100 I=1,N
      C(I)=WORK(I)
100   CONTINUE
      RETURN
      END
      SUBROUTINE PSRT1(LS,NS,C,CH)
      COMPLEX C(NS,2,LS),CH(2,NS,LS)
      DO 200 J=1,NS
      DO 200 I=1,LS
      CH(1,J,I)=C(J,1,I)
      CH(2,J,I)=C(J,2,I)
200   CONTINUE
      RETURN
      END
```

FIG. 5. Pease sort phase. 23 msec for 1024 complex input. Starts with serially ordered input. Loop inversion reduces this to 7 msec.

```
      SUBROUTINE PCMBN(IS,M,C,WORK)
C
C     COMBINE PHASE FOR THE PEASE FFT
C
      COMPLEX C(1),WORK(1)
      N=2**M
      DO 100 L=1,M
      LS=2**(L-1)
      NS=N/(LS+LS)
      CALL PCMB1(IS,LS,NS,C,WORK)
      DO 100 I=1,N
      C(I)=WORK(I)
100    CONTINUE
      RETURN
      END
      SUBROUTINE PCMB1(IS,LS,NS,C,CH)
      COMPLEX OMEGA,OMEGK,WYK,C(2,NS,LS),CH(NS,LS,2)
      ANGLE=FLOAT(IS)*4.*ATAN(1.)/FLOAT(LS)
      OMEGA=CMPLX(COS(ANGLE),SIN(ANGLE))
      OMEGK=1.
      DO 200 I=1,LS
      DO 100 J=1,NS
      WYK=OMEGK*C(2,J,I)
      CH(J,I,1)=C(1,J,I)+WYK
      CH(J,I,2)=C(1,J,I)-WYK
100    CONTINUE
      OMEGK=OMEGA*OMEGK
200    CONTINUE
      RETURN
      END
```

FIG. 6. Pease combine phase. Double subscripted. 34 msec for 1024 complex input. Follows sort phase.

```
      SUBROUTINE CANGLE(IS,N,NS2,OMEGK)
C
C     CANGLE INITIALIZES THE TRIGONOMETRIC TABLES FOR
C     THE PEASE FFT ALGORITHM
C
      COMPLEX OMEGK(NS2,M)
      N=2**M
      DO 100 L=1,M
      LS=2**(L-1)
      NS=N/(LS+LS)
      CALL INITP1(IS,LS,NS,OMEGK(1,L))
100   CONTINUE
      RETURN
      END
      SUBROUTINE INITP1(IS,LS,NS,OMEGK)
      COMPLEX OMEGA,OMEGK(NS,LS)
      ANGLE=FLOAT(IS)*4.*ATAN(1.)/FLOAT(LS)
      OMEGA=CMPLX(COS(ANGLE),SIN(ANGLE))
      OMEGH=1.
      DO 200 I=1,LS
      DO 300 J=1,NS
      OMEGK(J,I)=OMEGH
300   CONTINUE
      OMEGH=OMEGA*OMEGH
200   CONTINUE
      RETURN
      END
```

FIG. 7. Long-vector Pease FFT twiddle table generation. Less than 0.1 msec for 1024 complex input. Uses 1/2 N log2 N storage.

```
      SUBROUTINE PCMBNL(M,C,WORK,NS2,OMEGK)
C
C     COMBINE PHASE FOR LONG LOOP PEASE FFT;  NS2 = 2**(M-1)
C
      COMPLEX C(1),WORK(1),OMEGK(NS2,M)
      N=2**M
      DO 100 L=1,M
      CALL PCMBL1(NS2,OMEGK(1,L),C,WORK)
      DO 100 I=1,N
      C(I)=WORK(I)
100   CONTINUE
      RETURN
      END
      SUBROUTINE PCMBL1(NS2,OMEGK,C,CH)
      COMPLEX C(2,NS2),CH(NS2,2),OMEGK(1),WYK
      DO 200 JI=1,NS2
      WYK=OMEGK(JI)*C(2,JI)
      CH(JI,1)=C(1,JI)+WYK
      CH(JI,2)=C(1,JI)-WYK
200   CONTINUE
      RETURN
      END
```

FIG. 8. Long-vector Pease FFT combine phase. Single subscripted. 0.5 msec for 1024 complex input.

```
      SUBROUTINE STOCK(IS,M,C,WORK)
C
C     THE STOCKHAM AUTO-SORT FFT
C
      COMPLEX C(1),WORK(1)
      N=2**M
      DO 100 L=1,M
      LS=2**(L-1)
      NS=N/(LS+LS)
      CALL STOCK1(IS,LS,NS,C,WORK)
      DO 100 I=1,N
      C(I)=WORK(I)
100   CONTINUE
      RETURN
      END
      SUBROUTINE STOCK1(IS,LS,NS,C,CH)
      COMPLEX OMEGA,OMEGK,WYK,C(NS,2,LS),CH(NS,LS,2)
      ANGLE=FLOAT(IS)*4.*ATAN(1.)/FLOAT(LS)
      OMEGA=CMPLX(COS(ANGLE),SIN(ANGLE))
      DO 200 J=1,NS
      OMEGK=1.
      DO 200 I=1,LS
      WYK=OMEGK*C(J,2,I)
      CH(J,I,1)=C(J,1,I)+WYK
      CH(J,I,2)=C(J,1,I)-WYK
      OMEGK=OMEGA*OMEGK
200   CONTINUE
      RETURN
      END
```

FIG. 9. Stockham autosort FFT. 34 msec for 1024 complex input. Starts with serially ordered input, yields serially ordered output.

```
      SUBROUTINE STOCK(IS,M,C,WORK)
C
C     THE STOCKHAM AUTO-SORT FFT WITH LOOP INVERSION
C
      COMPLEX C(1),WORK(1)
      N=2**M
      DO 100 L=1,M
      LS=2**(L-1)
      NS=N/(LS+LS)
      CALL STOCK1(IS,LS,NS,C,WORK)
      DO 100 I=1,N
      C(I)=WORK(I)
100   CONTINUE
      RETURN
      END
      SUBROUTINE STOCK1(IS,LS,NS,C,CH)
      COMPLEX OMEGA,OMEGK,WYK,C(NS,2,LS),CH(NS,LS,2)
      ANGLE=FLOAT(IS)*4.*ATAN(1.)/FLOAT(LS)
      OMEGA=CMPLX(COS(ANGLE),SIN(ANGLE))
      IF(LS.LT.NS)GOTO 300
C$DIR SCALAR
      DO 200 J=1,NS
      OMEGK=1.
      DO 200 I=1,LS
      WYK=OMEGK*C(J,2,I)
      CH(J,I,1)=C(J,1,I)+WYK
      CH(J,I,2)=C(J,1,I)-WYK
      OMEGK=OMEGA*OMEGK
200   CONTINUE
      RETURN
300   OMEGK=1.
C$DIR SCALAR
      DO 400 I=1,LS
      DO 500 J=1,NS
      WYK=OMEGK*C(J,2,I)
      CH(J,I,1)=C(J,1,I)+WYK
      CH(J,I,2)=C(J,1,I)-WYK
500   CONTINUE
      OMEGK=OMEGA*OMEGK
400   CONTINUE
      RETURN
      END
```

FIG. 10. Stockham autosort FFT with loop inversion. 22 msec for 1024 complex input. Compiler directives were used to enforce the manual inversion.

## APPENDIX B

Convex execution times for the subroutines listed in Appendix A. All times are given in milliseconds per 1024 complex input vector. Fortran compilations were done with full vectorizing optimization. These times are for purposes of illustration; improvement is possible. In particular the sort routines might profitably be replaced by scatter-gather with precomputed indices. Also Swarztrauber reminds us that sorting is not needed when using FFT's for operations such as fast convolution where the user is not interested in seeing data in the frequency domain.

| Algorithm | Sort | Combine | Total |
|---|---|---|---|
| Cooley-Tukey | 25 | 40 | 65 |
| CT/loop invert | 25 | 18 | 43 |
| Gentleman-Sande | 24 | 40 | 64 |
| Pease | 23 | 34 | 57 |
| Pease/long vec | 23 | .5 | 24 |
| " + loop inv | 7 | .5 | 8 |
| Stockham | | | 34 |
| " + loop inv | | | 22 |

**MEMORANDUM**

**TO:** Publisher, The Journal of Irreproducible Results
P.O. Box 234
Chicago Heights, Ill. 60411
**FROM:** Dr. David M. Dean
Biology Department
University of South Alabama
Mobile, Ala. 36688
**COPY:** Dr. R.J. Beyers, Chairman, Biology Department
Dr. W.W. Kaempfer, Dean, Arts and Sciences
**RE:** Memoranda

It has occurred to me that the University runs entirely on memoranda, and if I spent as much time writing publications as I do memoranda, my vita would be huge. The obvious solution is to initiate a scientific journal that publishes memoranda.

J.A.M., the Journal of Academic Memoranda is hereby proposed to solve the problem of publish or perish. I have appointed myself editor and am very excited to announce to you that we will publish, in the first issue of J.A.M., a monograph of over seven years of memoranda from David M. Dean to various academic departments and personnel at U.S.A..

Unfortunately, we at the Journal have received the sad news that due to other commitments, our current editor will retire after the next issue. The staff at the Journal is actively seeking a new editor.

**COMMUNICATIONS**

The wonders of microcomputers and the capabilities of word processing and photocopying of documents have enabled rapid written communications even in emergency situations. Forms can be stored in the computer files, rapidly recalled and blanks filled in to generate memos immediately. Multiple copies can then be made on photo copying equipment and copies rapidly distributed. An example of a memo on file on floppy disc is reproduced below.

**To:** All Laboratory Personnel
**From:** W.A. Harada, M.D.
**Subject:** FIRE!
**Priority:** Urgent

**EASTON HOSPITAL**
**Department of Pathology**

**DATE:**_____
**TIME:**_____

1. It has been brought to my attention that a fire is raging in the _____ section of the laboratory.
2. All section supervisors will immediately review Laboratory Safety Manual, Section 3, FIRE, with personnel under their supervision.
3. Personnel previously assigned will deploy the CONFLAGRATION ATTENUATING APPARATUS from the usual wall storage devices and direct the nozzles in the general direction of the offending facilities.
4. Turn off all instruments including the LABORATORY BREWING APPARATUS in the Coffee Room.
5. Personnel not involved with suppressing the conflagration will abandon the premises shouting "FIRE" with maximum decibels.
6. Telephone the fire department. If there is no answer call 253-6304 (DUNKIN' DONUTS) where they probably are congregated at this hour.
7. Section supervisors will reply in writing acknowledgement of this memo through usual channels.

**W.A. Harada**
Easton, PA

cc: Administration
Engineering
Telephone Switchboard