OIT High Performance Computing Workshop

Performance Programming

on the

Cray YMP-EL

Dr. Andrew J. Pounds
High Performance Computing Group
Georgia Institutute of Technology

Performance Programming

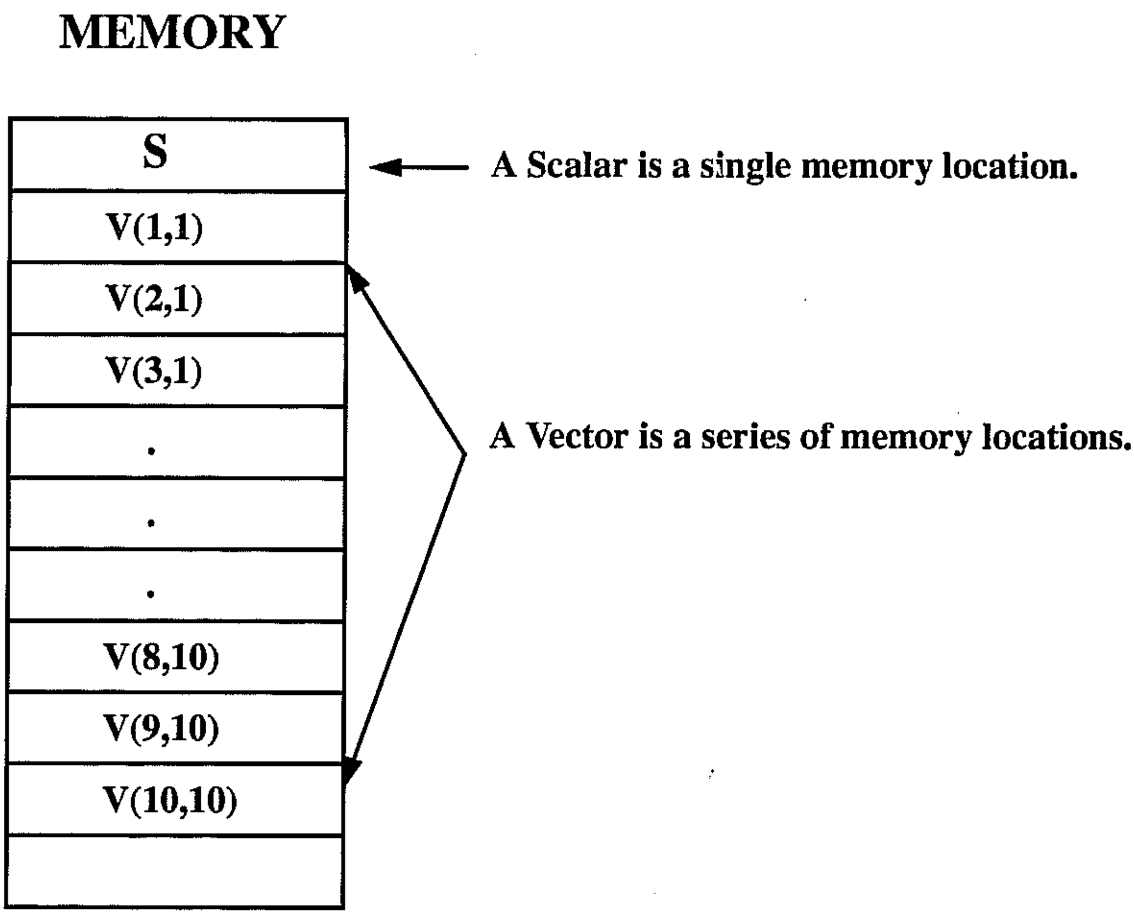
- **Vectorization** -- Processing chunks of contigous memory simultaneously rather than sequentially. (Optimizes CPU time)

- **Parallel Programming** -- breaking up a task over several processors. (Optimizes wall clock time)

Cray supercomputers have both of these performance enhancement capabilities. However, on a Cray YMP platform, vectorization is the critical issue in performance programming.

Our focus is vectorization!

What is a scalar and what is a vector?



Cray research systems, like other computing platforms, have machine instructions that operate on scalars. However, unlike most other computers, the Cray systems also have machine instructions that operate on vectors.

Why is Vector Processing Faster?

Example: Vector Multiply

```
      do 10 i = 1, N
         c(i) = a(i) * b(i)
10    continue
```

Scalar Processing:

```
1st operation:  c(1) = a(1) * b(1)
2nd operation:  c(2) = a(2) * b(2)
     .               .
     .               .
     .               .
Nth operation:  c(N) = a(N) * b(N)
```

Vector Processing

```
1st operation:  c(1) = a(1) * b(1)
                c(2) = a(2) * b(2)
                   .
                   .
                   .
                c(64) = a(64) * b(64)
2nd operation:  c(65) = a(65) * b(65)
                   .
                   .
                   .
                c(N) = a(N) * b(N)
```

VECTORIZATION IS MUCH FASTER!

OPERATION ORDER IS DIFFERENT USING VECTORIZATION

FORTRAN Code:

```
      DO 10 I = 1, 3
         L(I) = J(I) + K(I)
         N(I) = L(I) + M(I)
10    CONTINUE
```

Operations in Scalar Mode

Event	FORTRAN	Result
1	L(1)=J(1)+K(1)	7=2+5
2	N(1)=L(1)+M(1)	11=7+4
3	L(2)=J(2)+K(2)	-1=-4+3
4	N(2)=L(2)+M(2)	5=-1+6
5	L(3)=J(3)+K(3)	15=7+8
6	N(3)=L(3)+M(3)	15=15+0

Operations in Vector Mode

Event	FORTRAN	Result
1	L(1)=J(1)+K(1)	7=2+5
2	L(2)=J(2)+K(2)	-1=-4+3
3	L(3)=J(3)+K(3)	15=7+8
4	N(1)=L(1)+M(1)	11=7+4
5	N(2)=L(2)+M(2)	5=-1+6
6	N(3)=L(3)+M(3)	15=15+0

SO HOW DO YOU VECTORIZE CODE?

In order of importance:

- Use FPP (the Fortran Pre-Processor) to automatically vectorize the sections of code that it recognizes are vectorizable.

- Use optimized libraries (SCILIB, IMSL, etc.) whenever possible. This cannot can be stressed enough!

- Use the Cray Performance Analysis Tools to determine which routines are called the most and which routines use the most time.

- Once the most frequently called routines and the most time consuming routines have been located, manually insert vectorization directives and re-write sections as needed so that the compiler will vectorize them.

What will vectorize?

- Innermost loops addressing localized memory.

What will not vectorize?

- Outer loops
- Loops with I/O statements
- Loops with subroutine function calls
- Loops with data dependencies

KEY CONCEPT: Put as much computational work as possible into the innermost loops of your program.

Major Performance Analysis Tools

- **ja** -- (static) reports comprehensive statistics on time spent in program modes and memory usage.

usage:
```
$ja
$cf77 program.f
$a.out
$cut -c1-9,73-132 ja.out
$ja -st (terminates ja)
```

- **flowview/flowtrace** -- (dynamic) reports on which routines are called the most and are the best candidates for inlining.

usage:
```
$cf77 -F prog.f
$a.out
$flowview
```

- **jumpview/jumptrace** -- (dynamic) reports on which routines use the most time.

usage:
```
$cf77 -Wf"-ez" -ltrace prog.f
$jt a.out
$jumpview
```

Workshop Objectives

- Use of the Cray FORTRAN Pre-processor (FPP)
- Use of compilation listings to see how the pre-processor modifies code.
- Use of **ja** to get timing statistics
- Use of **flowview** and **jumpview** to determine which sections of code need work
- Coding strategies to obtain maximum vectorization

Logging in...

Log into the machines in the Baird Sun cluster using your prism account.

Type "`xterm`" and then in the new X-Window created by this command type "`xhost caracara`"

rlogin to caracara by typing the following command

```
rlogin caracara -l ccsupcc
```

The password is "`GTech#1`"

This ccsupcc account is running *csh*, so you will need to type the following command in order for the X-Windows utilities to function correctly.

```
setenv DISPLAY bairdsunX.gatech.edu:0.0
```

where the X is the number listed on your individual display.

Once you are logged into the account create yourself a directory, e.g. `mkdir joe`. Then change into this directory. Once inside your own directory, issue the following command:

`cp ../performance/*.f .` (the periods are important!)

You are now ready to begin the workshop!

Workshop Program #1: trigmat.f

- Lots of vector work in a single module.

Compile and run unvectorized

```
cf77 -Wf"-o novector -e mx" trigmat.f
ja
a.out
ja -c >ja.out
cut -c1-72 ja.out
ja -st
```

Record the run time (this is in the column "User CPU Seconds")

Compile Vectorized

```
cf77 -Wf"-e mx" trigmat.f
ja
a.out
ja -c >ja.out
cut -c1-72 ja.out
ja -st
```

Record the runtime.

Look at the Compilation Listing

```
vi trigmat.l
```

Compilation Listing... Notice the "V" vectorization loopmarks.

```
1TRIGMAT   PAGE 1      CRAY FORTRAN CFT77 6.0.3.0  02/11/94 15:12:12                                    02/0
6/95 10:08:45        PAGE 1
              L O O P M A R K   L E G E N D
              ----------------------------

              PRIMARY LOOP TYPE            LOOP MODIFIERS
              -----------------            --------------

              S - scalar loop              b - bottom loaded
              V - vector loop              c - computed safe vector length
              W - unwound loop             i - unconditionally vectorized with an IVDEP
              D - deleted loop             k - kernel scheduled
                                           r - unrolled
                                           s - short vector loop
                                           v - short safe vector length

1TRIGMAT   PAGE 1      CRAY FORTRAN CFT77 6.0.3.0  02/11/94 15:12:12                                    02/0
6/95 10:08:45        PAGE 2
                                           program trigmat

                                           parameter (maxdim = 500)

                                           real cosvec(maxdim), sinvec(maxdim), tanvec(maxdim)
                                           real mat1(maxdim,maxdim)
                                           real mat2(maxdim,maxdim)
                                           real mat3(maxdim,maxdim)

                                           x = 1.5

                                     ** Fill Vectors

              V----------------------<     do 10 i = 1, maxdim
              V                               cosvec(i) = cos(real(i)*x)
              V---------------------->10   continue

              V----------------------<     do 20 i = 1, maxdim
              V                               sinvec(i) = sin(real(i)*x)
              V---------------------->20   continue

              V----------------------<     do 30 i = 1, maxdim
              V                               tanvec(i) = sin(real(i)*x)/cos(real(i)*x)
              V---------------------->30   continue

                                     ** Fill Matrix 1

              S----------------------<     do 40 i = 1, maxdim
              S Vr-------------------<        do 50 j = 1, maxdim
              S Vr                               mat1(i,j) = cosvec(i) * sinvec(j)
              S Vr------------------->50      continue
              S---------------------->40   continue

                                     ** Fill Matrix 2

              S----------------------<     do 60 i = 1, maxdim
"trigmat.l" 244 lines, 17895 characters
```

Notice the Matrix Multiply Code...

```
 9.
10.                                   x = 1.5
11.
12.                               ** Fill Vectors
13.
14. V----------------------(      do 10 i = 1, maxdim
15. V                               cosvec(i) = cos(real(i)*x)
16. V---------------------->10    continue
17.
18. V----------------------(      do 20 i = 1, maxdim
19. V                               sinvec(i) = sin(real(i)*x)
20. V---------------------->20    continue
21.
22. V----------------------(      do 30 i = 1, maxdim
23. V                               tanvec(i) = sin(real(i)*x)/cos(real(i)*x)
24. V---------------------->30    continue
25.
26.                               ** Fill Matrix 1
27.
28. S----------------------(      do 40 i = 1, maxdim
29. S Vr-------------------(        do 50 j = 1, maxdim
30. S Vr                              mat1(i,j) = cosvec(i) * sinvec(j)
31. S Vr------------------->50      continue
32. S---------------------->40    continue
33.
34.                               ** Fill Matrix 2
35.
36. S----------------------(      do 60 i = 1, maxdim
37. S Vr-------------------(        do 70 j = 1, maxdim
38. S Vr                              mat2(i,j) = cosvec(i) * tanvec(j)
39. S Vr------------------->70      continue
40. S---------------------->60    continue
41.
42.                               ** Multiply Matrices
43.
44. S----------------------(      do 80 i = 1, maxdim
45. S S--------------------(        do 90 j = 1, maxdim
46. S S                               sum = 0.0
47. S S Vr-----------------(          do 100 k = 1, maxdim
48. S S Vr                              sum = sum + mat1(i,k)*mat2(k,j)
49. S S Vr----------------->100       continue
50. S S                               mat3(i,j) = sum
51. S S-------------------->90      continue
52. S---------------------->80    continue
1TRIGMAT   PAGE 2     CRAY FORTRAN CFT77 6.0.3.0  02/11/94 15:12:12                    02/0
6/95 10:08:45       PAGE 3

53.
54.                               ** Find maximum value in maatrix
55.
56.                                   rmax = mat3(1,1)
57.
58. S----------------------(      do 110 i = 1, maxdim
```

Compile with Additional Vector Preprocessing

```
cf77 -Z v -Wf"-e mx" trigmat.f
ja
a.out
ja -c >ja.out
cut -c1-72 ja.out
ja -st
```

Record the runtime.

Look at the Compilation Listing

```
vi trigmat.l
```

Notice how the pre-processor packed the loops. It turned loops 10, 20 and 30 into one loop!

```
TRIGMAT   PAGE 1      CRAY FORTRAN CFT77 6.0.3.0  02/11/94 15:12:12                                        02/0
6/95 10:58:15          PAGE 2
                                      program trigmat

                                      parameter (maxdim = 500)
                                C...Translated by FPP 6.0 (3.06E3) 02/06/95  10:58:12
dc
                                      real cosvec(maxdim), sinvec(maxdim), tanvec(maxdim)
                                      real mat1(maxdim,maxdim)
                                      real mat2(maxdim,maxdim)
                                      real mat3(maxdim,maxdim)

                                      REAL R1X, R2X
                                      x = 1.5

                                ** Fill Vectors

                                CDIR@ IVDEP
      V----------------------<        DO 10 I = 1, 500
      V                                  COSVEC(I) = COS(REAL@(I)*X)
      V                                  SINVEC(I) = SIN(REAL@(I)*X)
      V                                  TANVEC(I) = SIN(REAL@(I)*X)/COS(REAL@(I)*X)
      V---------------------->     10 CONTINUE

                                ** Fill Matrix 1

      S                         CDIR@    IVDEP
      S Vr-------------------<           DO J = 1, 500
      S Vr                                  MAT1(I,J) = COSVEC(I)*SINVEC(J)
      S Vr------------------->           END DO
      S---------------------->        END DO
      S----------------------<        DO I = 5, 500, 8
      S                         CDIR@    IVDEP
      S V--------------------<           DO 50 J = 1, 500
      S V                                   R1X = SINVEC(J)
      S V                                   MAT1(I,J) = COSVEC(I)*R1X
      S V                                   MAT1(1+I,J) = COSVEC(1+I)*R1X
      S V                                   MAT1(2+I,J) = COSVEC(2+I)*R1X
      S V                                   MAT1(3+I,J) = COSVEC(3+I)*R1X
      S V                                   MAT1(4+I,J) = COSVEC(4+I)*R1X
      S V                                   MAT1(5+I,J) = COSVEC(5+I)*R1X
      S V                                   MAT1(6+I,J) = COSVEC(6+I)*R1X
      S V                                   MAT1(7+I,J) = COSVEC(7+I)*R1X
      S V-------------------->     50    CONTINUE
      S---------------------->        END DO

                                ** Fill Matrix 2

      S----------------------<        DO I = 1, 4
      S                         CDIR@    IVDEP
      S Vr-------------------<           DO J = 1, 500
      S Vr                                  MAT2(I,J) = COSVEC(I)*TANVEC(J)
      S Vr------------------->           END DO
```

The pre-processor also replaced the matrix multiplication code with a *Cray Scientific Library* call. The pre-processor recognized the coding construct as matrix multiplication and replaced it with a more·efficient method!

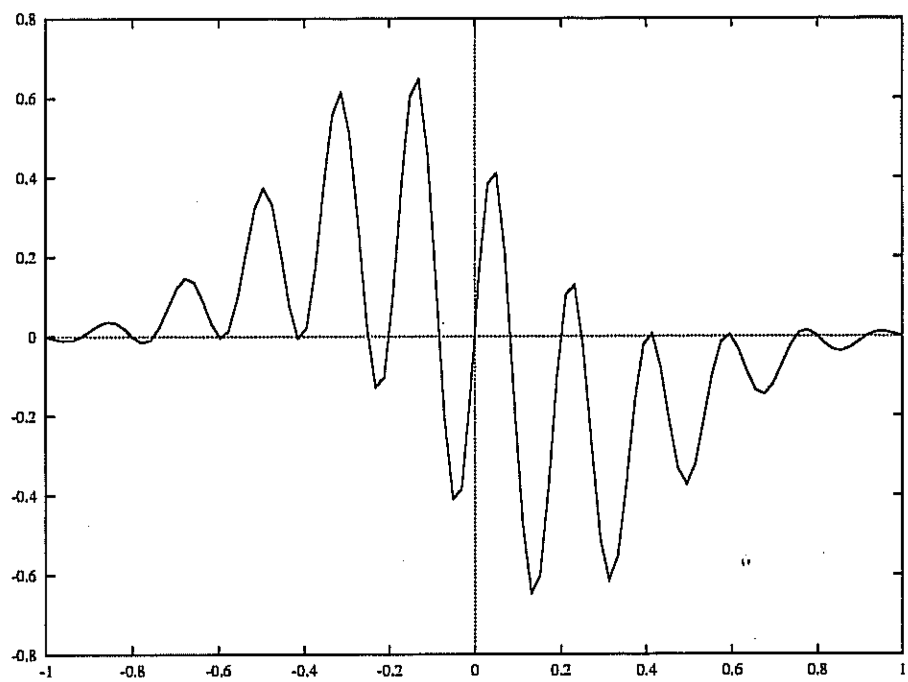
```
36. S V                                   MAT1(1+I,J) = COSVEC(1+I)*R1X
37. S V                                   MAT1(2+I,J) = COSVEC(2+I)*R1X
38. S V                                   MAT1(3+I,J) = COSVEC(3+1)*R1X
39. S V                                   MAT1(4+I,J) = COSVEC(4+I)*R1X
40. S V                                   MAT1(5+I,J) = COSVEC(5+I)*R1X
43. S V-------------------->   50     CONTINUE
44. S---------------------->       END DO
45.
46.                          ** Fill Matrix 2
47.
48. S----------------------(       DO I = 1, 4
49. S                        CDIR@    IVDEP
50. S Vr-------------------(          DO J = 1, 500
51. S Vr                                 MAT2(I,J) = COSVEC(I)*TANVEC(J)
52. S Vr------------------->          END DO
52. S Vr------------------->          END DO
1TRIGMAT   PAGE 2      CRAY FORTRAN CFT77 6.0.3.0  02/11/94 15:12:12                    02/0
6/95 10:58:15        PAGE 3

53. S---------------------->       END DO
54. S----------------------(       DO I = 5, 500, 8
55. S                        CDIR@    IVDEP
56. S V--------------------(          DO 70 J = 1, 500
57. S V                                  R2X = TANVEC(J)
58. S V                                  MAT2(I,J) = COSVEC(I)*R2X
59. S V                                  MAT2(1+I,J) = COSVEC(1+I)*R2X
60. S V                                  MAT2(2+I,J) = COSVEC(2+I)*R2X
61. S V                                  MAT2(3+I,J) = COSVEC(3+I)*R2X
62. S V                                  MAT2(4+I,J) = COSVEC(4+I)*R2X
63. S V                                  MAT2(5+I,J) = COSVEC(5+I)*R2X
64. S V                                  MAT2(6+I,J) = COSVEC(6+I)*R2X
65. S V                                  MAT2(7+I,J) = COSVEC(7+I)*R2X
66. S V-------------------->   70     CONTINUE
67. S---------------------->       END DO
68.
69.                          ** Multiply Matrices
70.
71.                                CALL SGEMMX@ (500, 500, 500, 1., MAT1(1,1), 1, 500, MAT2(1,1), 1,
72.                              1   500, 0., MAT3(1,1), 1, 500)
73.
74.                          ** Find maximum value in maatrix
75.
76.                                rmax = mat3(1,1)
77.
78.                          CDIR@ IVDEP
79. V----------------------(       DO 110 I = 1, 250000
80. V                                 RMAX = MAX@(RMAX,MAT3(I,1))
81. V---------------------->  110 CONTINUE
82.
83.                                print *, 'Maximum = ', rmax
84.
85.                                end
```

Workshop Program #2: ortho.f

- Functional Nightmare!

Program expands a function in a basis of Chebyshev polynomials. The function is:

$$f(x) = \cos(6\pi x)\sin(5\pi x)\exp\left(-4x^2\right)$$



The program is going to compute:

$$\chi_N^2 = \left[f(x) - \sum_{i=1}^{N}\left(\int_{-1}^{1} f(x)\,T_i(x)\,dx\right)T_i(x)\right]^2$$

For N=1 to 40, where T_i is the i[th] Chebyshev Polynomial

Compile with Additional Vector Preprocessing

```
cf77 -Z v -Wf"-e mx" ortho.f
ja
a.out
ja -c >ja.out
cut -c1-72 ja.out
ja -st
```

Record the runtime.

Look at the Compilation Listing

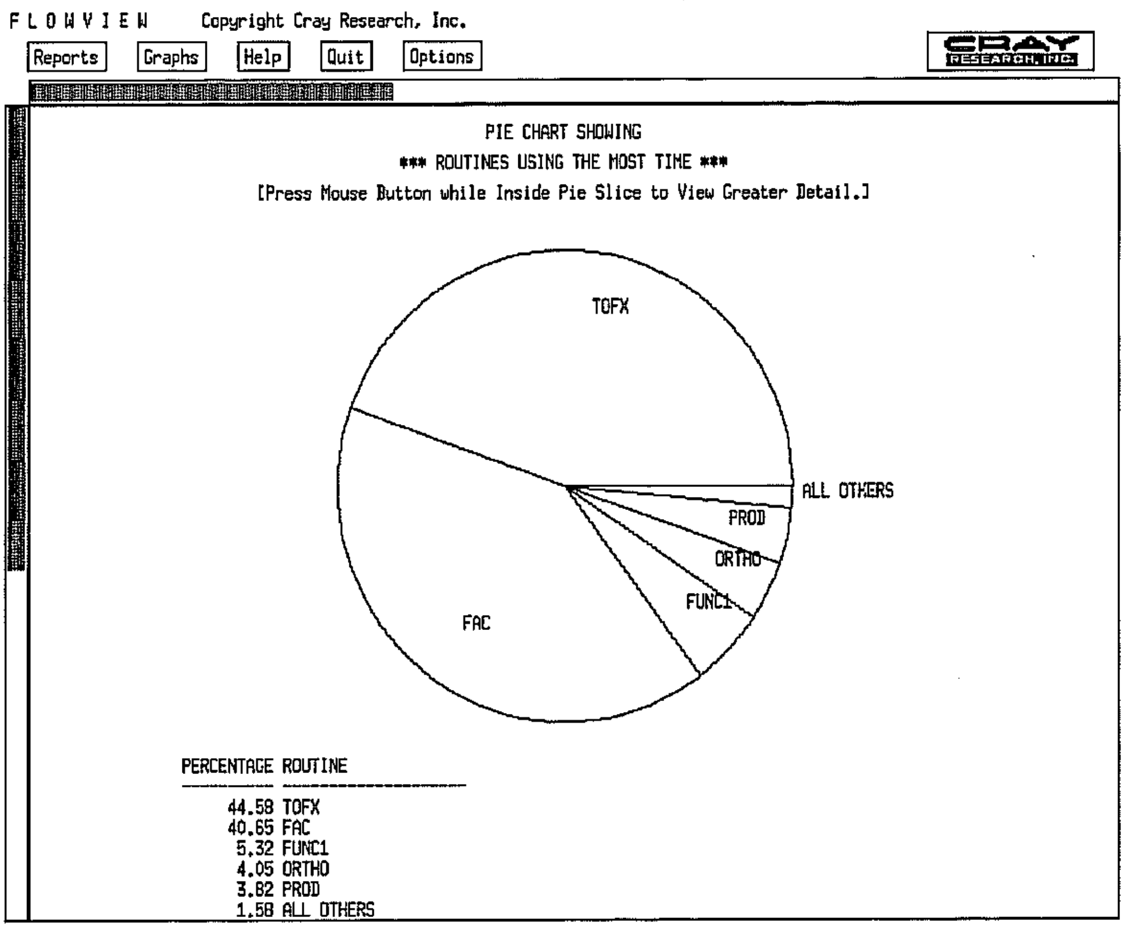
```
vi ortho.l
```

This code is characteristic of "old" code. The functions generally use iterative means to calculate values, and the code is not inlined at all. Because the program uses iterative means to evaluate functions, vectorization is not really helping us either because only the innermost loops are being vectorized. These typically do not have very long vector lengths and therefore the time spent in setting up the vector registers is essentially wasted. In fact, this code runs faster unvectorized! We will spend the rest of our time using the *Cray Performance Tools* to optimize this code.

The first thing we will do is reduce the number of program iterations by altering the parameter statement to `maxdim=10`. Although this slightly affects the statistics we will shortly be collecting, it greatly reduces the amount of time we will have to wait on the program to finish.

Now, compile for a **flowtrace** analysis and use **flowview**;

```
cf77 -F -Zv -Wf"-e mx" ortho.f
a.out
flowview
```



This shows that the Function `TofX` uses the most time. If you click on the pie region `TofX`, the following is displayed.

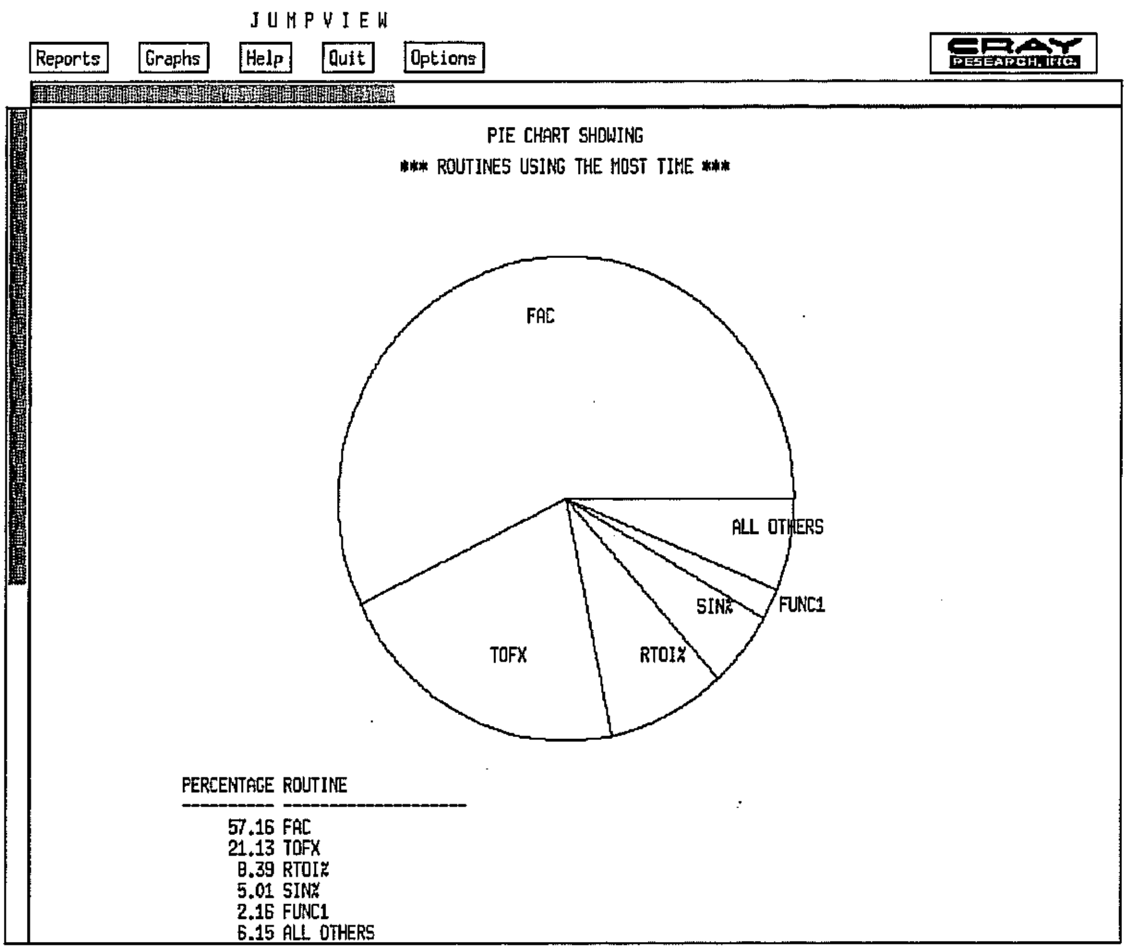
```
F L O W V I E W
[Reports] [Graphs] [Help] [Quit] [Options]                    CRAY RESEARCH, INC.

                    Flowtrace Statistics Report
                  Showing All Details For Routine
                               TOFX

 Routine was responsible for  44.6% of all program time.
 Routine used 2.59E+00 seconds of CPU time.
   or 86421103 clock periods.
 Routine was called 16335 times.
 Routine averaged 1.59E-04 seconds of CPU time per call.
   or 5291 clock periods per call.
 This routine is a candidate for in-lining,
   since its in-line factor is   3.1.
 Routine entry address: 561c

                    CALLED-BY TIMINGS FOR ROUTINE

Caller Name      Tot Time  # Calls Avg Time Routine Percentage
-----------      --------  ------- -------- ------- ----------
ORTHO            1.75E+00    11055 1.59E-04       67.64
PROD             8.39E-01     5280 1.59E-04       32.36
```

This tells us how much actual time was spent in this routine, including time to call (but not execute) other routines. If we want to learn about where the most CPU time was spent, we need to do a **jumptrace** analysis.

Now compile for a **jumptrace** analysis and use **jumpview.**

```
cf77 -Zv -Wf"-ez" -ltrace ortho.f
jt a.out
jumpview
```

The resulting X-Window display looks like the following:



Notice first that the routine PROD does not appear in the **jumptrace** analysis, but it does appear in the **flowtrace** analysis. Very little CPU time is spent inside the routine PROD, it basically calls two orther routines. Notice that jumpview also reports on SIN% and RTOI%, computationally intensive system routines. Click on the pie slice FAC.

Jumpview also gives you vectorization statistics, inlining statistics, and floating point performance.

```
                    J U M P V I E W
[Reports] [Graphs] [Help] [Quit] [Options]                    CRAY RESEARCH, INC.

                        JUMPTRACE DATA REPORT
               Showing Information About a Single Routine

Routine: FAC
 was responsible for  57.2% of total CPU time.

It was called   133650 times,
 and used 1.42E+00 seconds of CPU time,
    (47221218 clock periods).

Average CPU time used per call was 1.06E-05 seconds,
   (     354 clock periods).

This routine is a candidate for in-lining, since its in-line factor is 378.3.

The routine performed   1.3 million floating-point operations per second.

The routine's ratio of vector operations to scalar operations (all) is 2.09 : 1.

The routine's ratio of vector operations to scalar operations (floating-point) is  (infinite).

The routine's ratio of vector operations to scalar operations (memory) is 1.21 : 1.

The routine performed   3.3 million logical operations per second.

The routine performed   2.6 million memory operations per second.

The routine performed   1.5 vector operations per memory operation.

Total vector floating-point operations performed:
 Vector Floating-Point Add          521235
 Vector Floating-Point Multiply    1250964
 Vector Floating-Point Recip        104247
```

Clearly if we can increase the efficiency of `FAC` and `TofX`, we can dramatically improve our programs performance.

Currently the Chebyshev polynomial is calculated as follows:

$$T_n(x) = \frac{n}{2} \sum_{m=0}^{n/2} (-1)^m \frac{(n-m-1)!}{m!\,(n-2m)!} (2x)^{n-2m}$$

It can also be expressed as...

$$T_N(x) = \cos(N \operatorname{acos}(x))$$

This functional form removes any need to iterate (as in the present form) and also eliminates the need to calculate factorials. Replace the appropriate lines in the `TofX` function, change `maxdim` back to 40, recompile and compare your timing results.

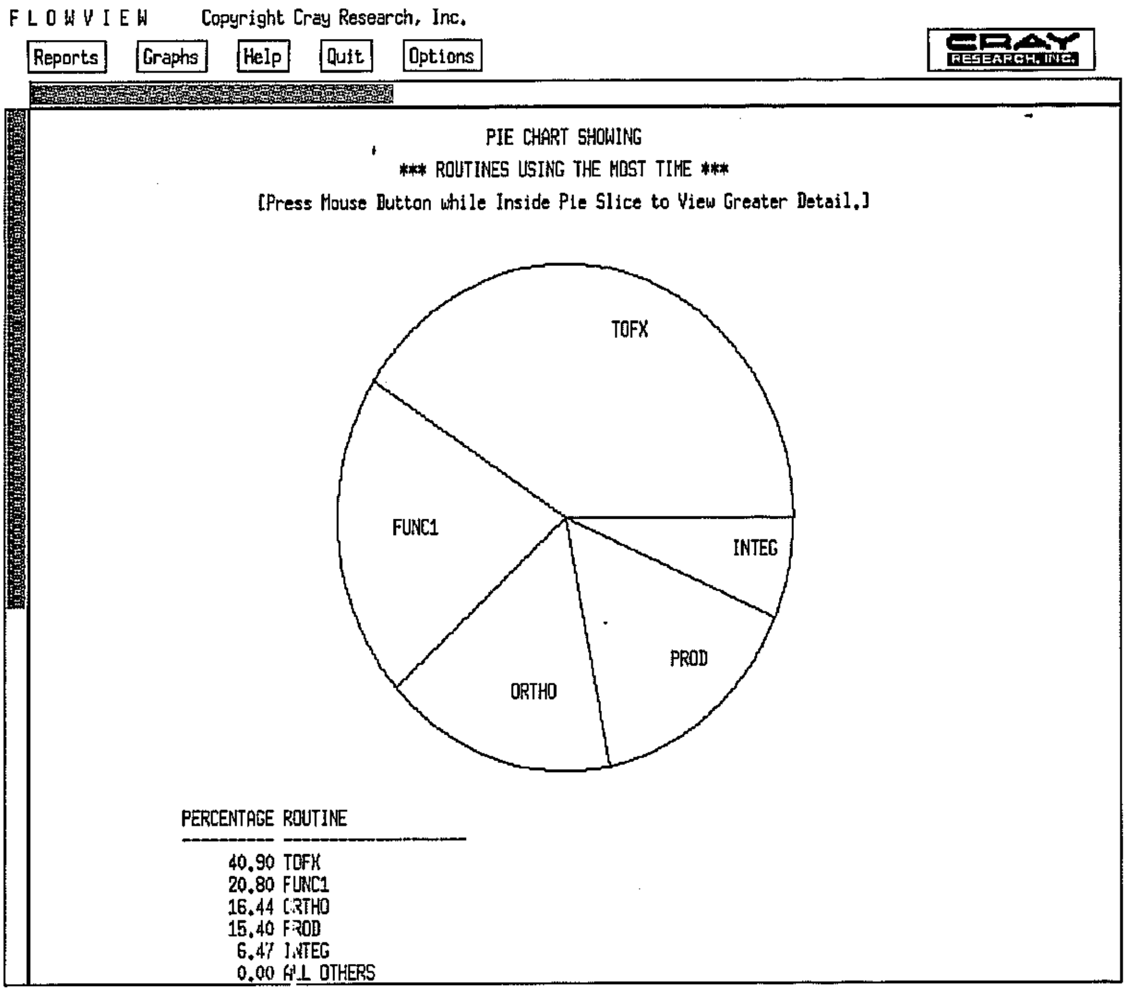
```
cf77 -Zv -Wf"-e mx" ortho.f
ja
a.out
ja -c >ja.out
cut -c1-72 ja.out
ja -st
```

Record your timing results.

Now, change the maxdim statement back to 10 and recompile for a **flowtrace** analysis.

```
cf77 -F -Zv -Wf"-e mx" ortho.f
a.out
flowview
```

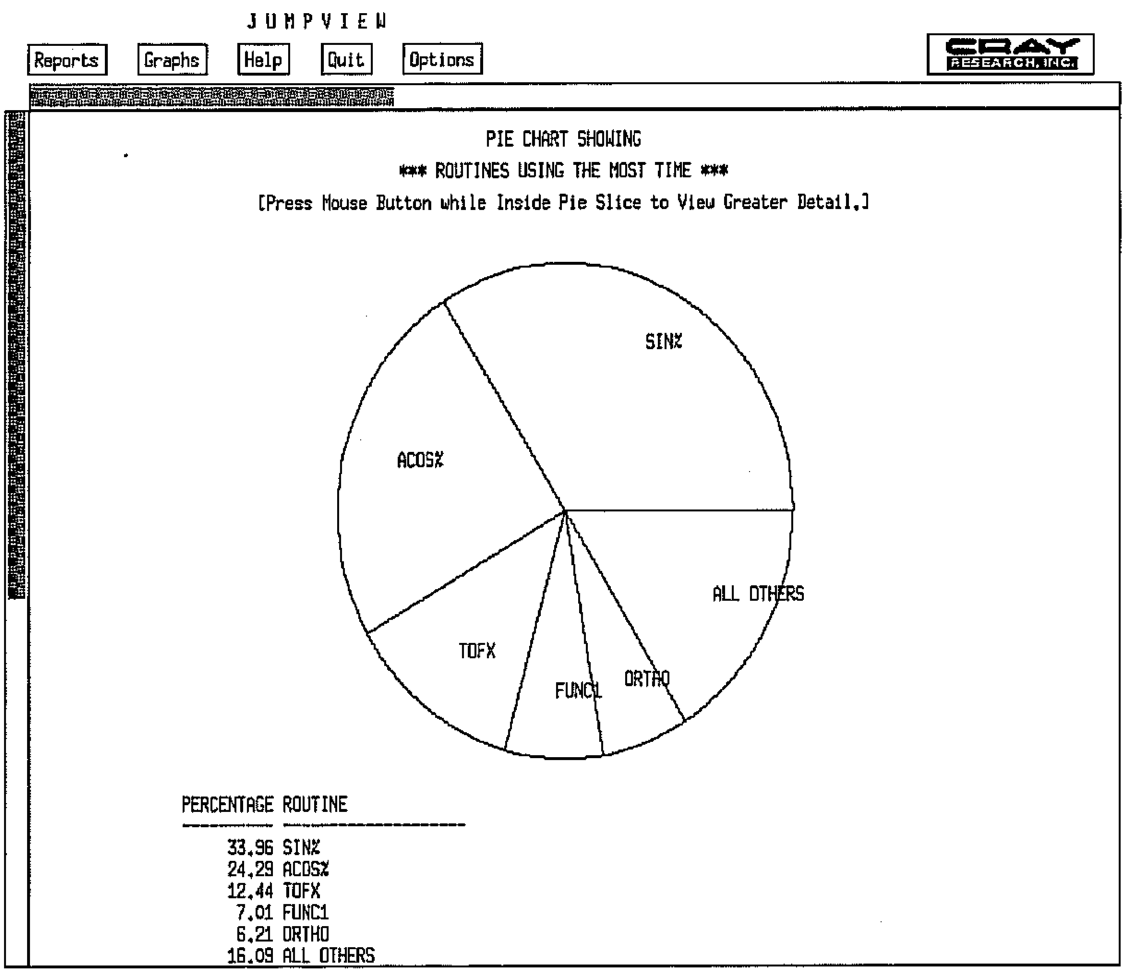
Here is the resulting **flowview** analsysis:



Similarly, prepare a **jumptrace** analysis

```
cf77 -Zv -Wf"-ez" -ltrace ortho.f
jt a.out
jumpview
```

The resulting **jumpview** analysis looks like the following:



Notice first of all that the routine `FAC` is no longer present. From the **Flowview** analysis, it is clear that `TofX`, `FUNC1` and `PROD` are the routines to optimize. The **Jumpview** analysis tells us that `TofX` and `FUNC1` are doing most of the computational work now (because they contain the `SIN%` and `ACOS%` functions). The time `PROD` is using to set up and make the two function calls is essentially wasted.

Now we need to work on coding strategies:

Optimization Techniques handout...

Your mission:

I have gotten this code to run (with 40 basis functions) in under 1 second. See if you can match or beat this time. Here are three hints:

1. Inlining.
2. Use the compilation listings to determine what loops are being vectorized and which are not.
3. Where is there a big loop, with lots of work in it, that could be vectorized?

You are welcome to copy the programs to you own account if you would like to continue to experiment with them. They are readable in the directory ~ccsupcc/performance