CRAY RESEARCH, INC.

CRAY® COMPUTER SYSTEMS

COS PERFORMANCE UTILITIES
REFERENCE MANUAL
SR-0146

RECORD OF REVISION CRAY RESEARCH, INC. PUBLICATION NUMBER SR-0146

Each time this manual is revised and reprinted, all changes issued against the previous version are incorporated into the new version and the new version is assigned an alphabetic level.

Every page changed by a reprint with revision has the revision level in the lower righthand corner. Changes to part of a page are noted by a change bar in the margin directly opposite the change. A change bar in the margin opposite the page number indicates that the entire page is new. If the manual is rewritten, the revision level changes but the manual does not contain change bars.

Requests for copies of Cray Research, Inc. publications should be directed to the Distribution Center and comments about these publications should be directed to:

CRAY RESEARCH, INC.
1345 Northland Drive
Mendota Heights, Minnesota 55120

Revision	Description
	April 1987 - Original printing.

PREFACE

This manual describes performance utilities that run on the CRAY X-MP and CRAY-1 computer systems under the Cray operating system COS. They can be used with at least one of the following language processors:

- Cray Fortran compiler CFT77
- Cray Fortran compiler CFT
- Cray Pascal compiler
- Cray C compiler
- Cray Assembly Language (CAL) assembler

This manual describes the means of invoking these utilities, their options, and their output.

Related publications from Cray Research, Inc. (CRI) are the following:

SR-0018	CFT77 Reference Manual
SR-0009	Fortran (CFT) Reference Manual
SR-0222	CRAY X-MP Multitasking Programmer's Manual
SR-2024	Cray C Reference Manual
SR-0060	Pascal Reference Manual
SR-2003	CAL Assembler Version 2 Reference Manual
SR-0011	COS Version 1 Reference Manual
SR-0112	Symbolic Debugging Package Reference Manual

Within this manual, all publication references refer to CRI publications.

CONVENTIONS

This manual uses the following conventions:

Convention	Description
italics	Indicates variable information usually specified by the user or terms being defined
[ ]	Brackets enclose optional portions of a command or directive format
bold	Denotes a word from an output listing spelled as in the listing

READER COMMENTS

If you have any comments about the technical accuracy, content, or organization of this manual, please tell us. You have several options that you can use to notify us:

- Call our Technical Publications department directly at (612) 681-5729 during normal business hours

- Send us UNICOS or UNIX electronic mail at:

  ihnp4!cray!publications or **sun!tundra!hall!publications**

- Use the Reader Comment form at the back of this manual

- Write to us at the following address:

  Cray Research, Inc.
  Technical Publications Department
  1345 Northland Drive
  Mendota Heights, Minnesota 55120

We value your comments and assure a prompt response.

CONTENTS

INTRODUCTION 1

This manual documents COS utilities for analyzing the performance of application programs.

FTREF statically analyzes Fortran source code as an aid in understanding program structure, keeping track of variables and common blocks, and other aspects of programming. It is especially useful in the analysis of programs for multitasking. FTREF output includes a summary of common block references, a summary of entries and subprogram calls, and a static calling tree.

FLOWTRACE monitors calls to and from all program units during execution of a program and prints a summary of these calls with statistics about execution time. By including in a program unit's timing the time used by its libary calls, it indicates the program unit's total execution time. FLOWTRACE output includes a summary of subprogram calls and timings and a calling tree. SETPLIMQ can be used to give separate timings for each call.

SPY gives a statistical picture of the time spent in specific areas of code by monitoring the program address register during execution. The granularity of code areas to be monitored is selected by the user, and can be as small as a single word of instructions. This fine-grained monitoring differs from FLOWTRACE's results at the level of program units. Although SPY's results are approximate, they are useful for isolating inefficient code. SPY output includes a line for each specified code area, or *bucket*. Each line shows statistics on how many times one bucket was accessed, shown in the following ways:

- An absolute count of accesses of the address bucket
- The bucket's percentage of accesses within the routine
- The bucket's percentage of accesses within the program

Another table summarizes these results for entire routines rather than individual buckets.

PERFMON monitors hardware performance during program execution and can be used for further analysis of execution speed. PERFMON output includes detailed information concerning scalar activity, hold issue conditions, memory use, or vectorization.

This manual uses the terms *program unit* and *subprogram* in their Fortran senses, as defined in the reference manuals for the Cray Fortran compilers and by the American National Standards Institute (ANSI) standard X3.9-1978, often called Fortran 77. For Fortran programs, both terms refer to Fortran code that you compile, as distinguished from library routines or procedures written in other languages. A *program unit* can be a main program or a subprogram. A *subprogram* is either a subroutine, function, or block data subprogram.

In other languages the term *subprogram* is equivalent to the following terms:

- Pascal: procedure or function
- C: function
- CAL: entry

FTREF - ANALYZING FORTRAN SOURCE CODE 2

The FTREF utility analyzes Fortran source listings and provides structural information about it. This analysis does not use information derived from executing a program. The analysis contains the following information:

- Global cross-reference listing of common block variables
- A static calling tree showing entry names, calling program units, and called subprograms for each subprogram
- For multitasked applications, a summary of macrotasking subroutines used, with options to indicate whether a common variable or a subroutine is locked when it is referenced or redefined

FTREF processes listings generated by either CFT77 or CFT. (CFT output should not be mixed with CFT77 output, however.) The dataset to be processed by FTREF may contain any number of program units used by the application. The more program units that are included in the dataset, the more complete will be the information output by FTREF. A listing can also contain output from multiple compilations using different levels of a compiler.

FTREF is invoked by the FTREF control statement. A Fortran job must have been compiled previously, with the Fortran control statement specifying ON=XS to generate a cross-reference table and a source listing. The Fortran control statement must have specified a listing dataset with L=, and the FTREF control statement specifies an input dataset with I=. These should match unless you need to use different datasets, such as to combine more than one Fortran compilation into one dataset.

The following examples show how to invoke FTREF in the same job as the compile step. These steps can be done in different jobs.

JCL examples:

```
JOB,...
ACCOUNT,...
CFT77,ON=SX,L=LISTX.
FTREF,I=LISTX,CB=FULL,TREE=FULL.
/EOF
    CFT77 source
/EOF
```

```
JOB,...
ACCOUNT,...
CFT,ON=SX,L=LISTX.
FTREF,I=LISTX,CB=FULL,TREE=FULL.
/EOF
    CFT source
/EOF
```

In the preceding examples, the compiler takes source code from the next file in the dataset and writes the source listing and cross-reference table in dataset LISTX; FTREF then uses LISTX as its input. The CB=FULL and TREE=FULL options represent a typical basic use of FTREF.

2.1 FTREF CONTROL STATEMENT

The FTREF control statement invokes FTREF to analyze listings from a previous Fortran compilation. The control statement can be in the same job as the compilation, as shown in the previous examples, or in a different job. The I parameter is required.

Format:

```
FTREF,I=idn,L=ldn,CB=op,TREE=op,ROOT=r,END=end,

  LEVEL=n,DIR=dir,NORDER,MULTI.
```

I=*idn*	Input dataset containing listings generated by one or more Fortran compilations, in which the ON=SX options were specified. This parameter is required. Unless different datasets are needed, the I specification agrees with the L= dataset shown on the Fortran control statement.
L=*ldn*	Dataset to receive the output listing; the default is $OUT.
CB=*op*	Global common-block cross-references. The options are as follows:

PART	Identifies program units that use each common block; this is the default.
FULL	Details the use of common block variables in all program units.
NONE	No common block output

TREE=*op* Produces the static calling tree for the program and information about the subprograms called. The value **LOOP** in the output indicates there is an apparent recursive call in the program. The options are as follows:

- PART For each program unit, reports the entry names, external calls, names of program units that call it, and common block names; this is the default.
- FULL Reports the information that the PART option provides plus the static calling tree. The ROOT, END, and LEVEL parameters and the ROOT directive can be used to manage the calling tree output. See the examples at the end of this section.
- NONE No output information

ROOT=*r* If TREE=FULL, ROOT causes subprogram *r* to be the root of the tree. Use ROOT when you want to see only one subtree of the calling tree. Use the ROOT directive (see subsection 2.2.1) if you want to see more than one subtree. The default root is the main program; that is, the program unit that is not called. If there is more than one such program unit, the first by alphabetic order is used.

END=*end* If TREE=FULL, END directs subprogram *end* to terminate any branch of the tree in which *end* is encountered. The value **STOP** is printed whenever the *end* subprogram is found, and that branch of the tree is terminated. By default, any calling tree generated by FTREF contains all subprograms in the program.

LEVEL=*n* If TREE=FULL, LEVEL limits the length of any branch to *n* levels deep. The default is the entire program. If both LEVEL and END are specified, FTREF terminates a branch of the tree at whichever state is encountered first.

DIR=*dir* Dataset *dir* contains a set of directives that control the processing (see subsection 2.2). If DIR is specified without a value, the directives are taken from $IN, whether the Fortran source is in $IN or another dataset. If DIR is not specified, no directives are read.

NORDER Lists the source program's subprograms in input order instead of alphabetic order. The default is alphabetic order.

MULTI Examines the source for uses of the macrotasking subroutines and generates tables summarizing the subroutines' use within the program. See the CRAY X-MP Multitasking Programmer's Manual, publication SR-0222.

2.2 DIRECTIVES

FTREF directives are placed in a separate file; see the description of the DIR option on the FTREF control statement.

A directive consists of a keyword and zero or more parameters. A blank, comma, or opening parenthesis separates the keyword and the parameters. The terminator is a period, closing parenthesis, or two consecutive blanks.

A caret at the end of the directive line indicates that the next line is a continuation of the current directive. Do not precede the caret by a blank; it must immediately follow the last nonblank character of the line.

Each line should contain only one directive. FTREF processes 80 characters per line and ignores the rest of the characters.

Examples:

The following examples show the use of FTREF with a directive file. In both examples, FTREF is invoked in the same job with the compile step. The JCL, source file, and directive file are all in $IN. The DIR keyword on the FTREF control statement causes a directive file to be read from $IN (in these cases, the file following the source file). This file contains the CHKBLK directive.

```
JOB,...
ACCOUNT,...
CFT77,ON=SX,L=SOURCE.
FTREF,I=SOURCE,CB=FULL,TREE=FULL,DIR.
/EOF
      CFT77 source
/EOF
CHKBLK
/EOF
```

```
JOB,...
ACCOUNT,...
CFT,ON=XS,L=SOURCE.
FTREF,I=SOURCE,CB=FULL,TREE=FULL,DIR.
/EOF
      CFT source
/EOF
CHKBLK
/EOF
```

2.2.1 ROOT DIRECTIVE

The ROOT directive specifies a subprogram to be used as the root of a static calling tree (TREE=FULL). The output for each subprogram (specified in this directive or on the ROOT parameter) is a separate tree. Default processing is described with the ROOT parameter.

ROOT,md_1,md_2,...md_n.

md_n	Name of a root subprogram

2.2.2 SUBSET DIRECTIVE

The SUBSET directive specifies which program units will be processed by FTREF and prevents FTREF from processing program units not on the parameter list. If no SUBSET directive appears, or only one SUBSET directive is specified and it has no parameters, FTREF processes all program units in the input file.

SUBSET,md_1,md_2,...md_n.

md_i	Name of the program unit to be processed by FTREF

2.2.3 CHKBLK DIRECTIVE

The CHKBLK directive specifies the common blocks whose variables FTREF will check to determine if the variables appear only in locked areas in a multitasked program. If no CHKBLK directive appears, FTREF does not check on whether variables are in a locked area. If there is only one CHKBLK directive and it is without a parameter, FTREF processes all the common block variables.

A locked area begins with a LOCKON call and ends with a LOCKOFF call, both with the same lock variable. If a common block variable is referenced or redefined in such an area, it is considered to be locked; otherwise, it is unlocked. If a variable reference is unlocked, check to ensure that this is a safe condition. A locked area must be completely within a program unit to be detected by FTREF. Refer to the CRAY X-MP Multitasking Programmer's Manual, publication SR-0222, concerning the locking feature.

```
CHKBLK,blk1,blk2,...blkn.
```

blk_i Name of the common block whose variables FTREF will check to determine if a variable is in a locked area. To specify blank common, use // .

NOTE

FTREF searches for the presence of the LOCKON and LOCKOFF calls. It does not attempt a flow analysis; that is, it does not consider the effects of IF or GOTO statements on locks or variables.

2.3 OUTPUT EXAMPLE

In the following example of FTREF output, longer lines have been truncated to fit the page. The global common block reference is an excerpt of longer output.

```
                    ***********************************************
                    *                                             *
                    *     SUMMARY OF COMMON BLOCK REFERENCES      *
                    *                                             *
                    ***********************************************

  BLOCK NAME
  #DA       IS USED BY:
                         BKSUB     DIFEQ     PINVS     PLGNDR  RED    SFROID    SOLVDE
  //        IS USED BY:
                         DIFEQ     SFROID
  SMATRIX   IS USED BY:
                         DIFEQ     PINVS     RED
```

```
                *******************************************
                *                                         *
                *    GLOBAL COMMON BLOCK CROSS REFERENCE  *
                *                                         *
                *******************************************

 BLOCK NAME
 #DA
 ADDRESS NAME     TYPE MAIN USAGE   MODULE         SOURCE PROGRAM REFERENCE
     246 C         R   3DIM ARRAY   SFROID           52P     4D

    1041 CONV      R   VARIABLE     SFROID           52P     9S

    1044 I         I   VARIABLE     SFROID           28U/2  27I

      50 INDEXV    I   1DIM ARRAY   SFROID           52P    22S     21S     20S
      50 NVARS     I   VARIABLE     SOLVDE           58U     8S

//
 ADDRESS NAME     TYPE MAIN USAGE   MODULE         SOURCE PROGRAM REFERENCE
      55 ANORM     R   VARIABLE     SFROID           46P    45P     42U
      55 ANORM     R   VARIABLE     DIFEQ            26U     3D

      54 C2        R   VARIABLE     SFROID           53U    51U     50S
      54 C2        R   VARIABLE     DIFEQ            35U    22U     19U

      51 H         R   VARIABLE     SFROID           33U    11S      3D
      51 H         R   VARIABLE     DIFEQ            48U    34U     32U

      52 MM        I   VARIABLE     SFROID           53U/3  44U     43U/2
                                                     27N    25U     15U
      52 MM        I   VARIABLE     DIFEQ            49U    37U     22U

SMATRIX
 ADDRESS NAME     TYPE MAIN USAGE   MODULE         SOURCE PROGRAM REFERENCE
       0 S         R   2DIM ARRAY   PINVS          64U     54S     52U/2   52S
       0 S         R   2DIM ARRAY   RED            15U/2   15S     10U/2   10S
       0 S         R   2DIM ARRAY   DIFEQ          50S     49S     48S     47S


        ABBREVIATION USED ABOVE (KEY TO THE LINE NUMBER)
        ------------------------------------------------
        D  DEFINED IN DECLARATIVE STATEMENT
        N  NAME USED AS A DO LOOP PARAMETER
        P  ARGUMENT IN CALL OR FUNCTION CALL
        S  STORED SO CONTENTS MAY BE CHANGED
        U  NAME USED IN EXECUTABLE STATEMENT
        *  DEFINED OR DECLARED BUT NEVER USED
        /2 REFERENCED MORE THAN ONE TIME IN THE STATEMENT
```

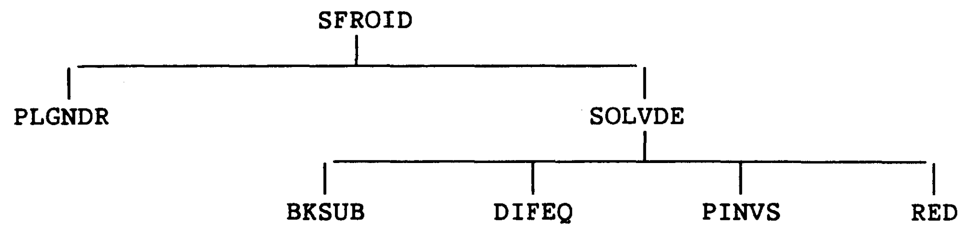
```
               *************************************************
               *                                               *
               *     SUMMARY OF ENTRIES AND ROUTINE CALLS      *
               *                                               *
               *************************************************
BKSUB
               ENTRY : BKSUB
               CALLS : ** NONE **
           CALLED BY : SOLVDE
           BLOCK USED: #DA
DIFEQ
               ENTRY : DIFEQ
               CALLS : ** NONE **
           CALLED BY : SOLVDE
           BLOCK USED: #DA      //       SMATRIX
PINVS
               ENTRY : PINVS
               CALLS : ** NONE **
           CALLED BY : SOLVDE
           BLOCK USED: #DA      SMATRIX
PLGNDR
               ENTRY : PLGNDR
               CALLS : ** NONE **
           CALLED BY : SFROID
           BLOCK USED: #DA
RED
               ENTRY : RED
               CALLS : ** NONE **
           CALLED BY : SOLVDE
           BLOCK USED: #DA      SMATRIX
SFROID
               ENTRY : SFROID
               CALLS : PLGNDR   SOLVDE
           CALLED BY : ** NONE **
           BLOCK USED: #DA      //
SOLVDE
               ENTRY : SOLVDE
               CALLS : BKSUB    DIFEQ    PINVS    RED
           CALLED BY : SFROID
           BLOCK USED: #DA
```

```
                    *********************************************
                    *                                           *
                    *      STATIC CALLING TREE FOR SFROID       *
                    *                                           *
                    *********************************************

SFROID    PLGNDR
 < 1 >    SOLVDE    BKSUB
           < 2 >    DIFEQ
           < 2 >    PINVS
           < 2 >    RED
                     ****** END OF THE CALLING TREE ******
```

Notes on the preceding output

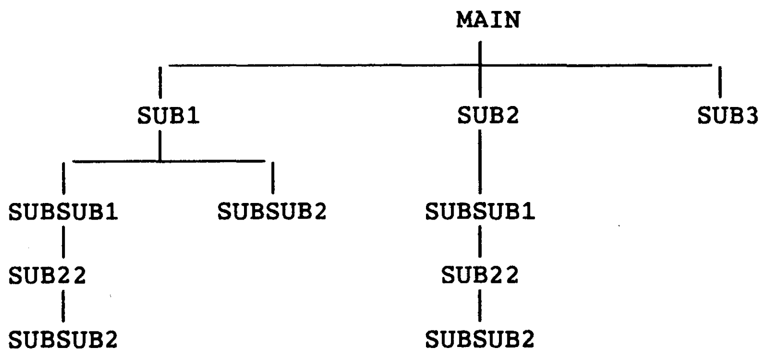
The preceding calling tree would look as follows in graphic form:



Another calling tree example:

```
MAIN      SUB1      SUBSUB1   SUB22     SUBSUB2
          < 2 >     SUBSUB2
< 1 >     SUB2      SUBSUB1   SUB22     SUBSUB2
< 1 >     SUB3
```

The above calling tree would look as follows in graphic form:



FLOWTRACE - TRACING SUBPROGRAM CALLS 3

FLOWTRACE generates printed information about all subprogram calls in a program; the output is based on execution of the program. The summary is written to dataset $OUT. The following information is shown:

- The time spent in each program unit: amount, percentage of the total execution time, and average time per call
- Number of calls to each subprogram and total calls in the program
- Lists of subprograms that call and are called by each subprogram
- A calling tree of the main program and all subprograms

3.1 ENABLING AND DISABLING FLOWTRACE

Flowtracing is activated by compiler options or directives, as follows:

CFT77 or CFT:
The main program must begin with a PROGRAM statement. Flowtracing can be enabled for an entire program or for parts of a program, as follows:

- To trace an entire program, include ON=F on the CFT77 or CFT control statement.
- To trace specific parts of a program, insert directives in the source code: CDIR$ FLOW enables, and CDIR$ NOFLOW disables. In CFT77, these directives must be within a program unit. In CFT, FLOW must precede a program unit's header statement, and NOFLOW must follow its END statement.

C: The -F option on the CC control statement applies to the whole executable program. See the following text concerning libraries.

Pascal: The BT+ option can appear on the PASCAL control statement to apply to the whole program, or within compiler directives to enable flowtracing for specific parts of a program. Used within compiler directives, BT+ is disabled by BT-. See the following text concerning libraries.

When you run FLOWTRACE with C or Pascal, you must use the STACK or MULTI versions of libraries to be accessed. The default version of the libraries is site-dependent, as is the manner of selecting another version. Consult your site analyst about the default and the method of selection.

It may be useful to disable flowtracing for small, frequently called subprograms because the FLOWTRACE overhead time can exceed the execution time.

When flowtracing is enabled, a FLOWTRACE summary is listed after execution. Programs that terminate with library calls such as EXIT or ABORT must be modified for flowtracing: these calls should be replaced by END or STOP statements.

Time spent in a lower-level called subprogram for which flowtracing is enabled is not counted as time spent in the calling subprogram. Time spent in library routines (such as SIN, PRINT, or CFFT) or in any subprogram for which flowtracing is not enabled is counted as time spent in the calling subprogram; these called subprograms are not listed in the summary.

The output headed "Alphabetized summary" shows the following information:

Column	Description
(leftmost)	Index into the calling tree for the first 999 subprograms; *** indicates subsequent subprograms.
Routine	Subprogram name, such as the name in a FUNCTION or SUBROUTINE statement
Time executing	Seconds of CPU time and percentage of the program's total time. The > symbol indicates a time less than 1 ms; >>> means less than 1 microsecond.
Called	Number of calls to the subprogram
Average T	Average time per call to the subprogram. The > symbol indicates a time less than 1 ms; >>> means less than 1 microsecond.
(next right)	Absolute address of the subprogram
(rightmost)	Calling subprograms. If nothing follows the phrase "Called by," the subprogram was not called; normally this indicates the main program or the system. See the line for SFROID in the following example.

FLOWTRACE displays only the first 28 routines that called a given routine. If you require more than 28, ask your site analyst to modify the source for FLOWTRACE in $UTLIB, and rebuild it for you.

Example (longer lines truncated):

```
F L O W T R A C E  --  Alphabetized summary

   Routine      Time executing  Called  Average T
 7 BKSUB     0.007    ( 12.49%)     15            > @00003361a Called by SOLVDE
 4 DIFEQ     0.002    (  3.47%)    630            > @00004717a Called by SOLVDE
 5 PINVS     0.029    ( 53.21%)    630            > @00004044a Called by SOLVDE
 2 PLGNDR        >    (  0.67%)    120            > @00002340a Called by SFROID
 6 RED       0.007    ( 13.30%)    615            > @00004431a Called by SOLVDE
 1 SFROID    0.002    (  3.50%)      1        0.002 @00002012a Called by
 3 SOLVDE    0.007    ( 13.36%)      4        0.002 @00003006a Called by SFROID
   TOTAL     0.054                2015 Total calls
```

The output headed "Calling tree" displays the dynamic calling tree by progressively indenting each successive level of the tree.

Example:

```
F L O W T R A C E  --  Calling tree

    1   SFROID   00002012a
    2     PLGNDR   00002340a
    3     SOLVDE   00003006a
    4       DIFEQ    00004717a
    5       PINVS    00004044a
    6       RED      00004431a
    7       BKSUB    00003361a
```

The above tree is shown graphically at the end of section 2.

3.2 FLODUMP UTILITY: FOR ABORT CONDITIONS

The FLODUMP utility recovers and dumps FLOWTRACE tables when a program aborts. It is activated by the FLODUMP control statement in the program's JCL, and should be included only if flowtracing is enabled (see subsection 3.1). FLODUMP also requires a previous DUMPJOB statement. The following example illustrates the use of the FLODUMP control statement:

```
JOB, ... .
CFT77,ON=SF.
SEGLDR,GO.
EXIT.
DUMPJOB.
FLODUMP.
...
```

Format:

```
FLODUMP[,L=ldn].
```

L=*ldn*	Dataset to write the report to; the default is $OUT.

3.3 SETPLIMQ SUBROUTINE

Subroutine SETPLIMQ enables FLOWTRACE to print a line on the output listing file for every CALL or RETURN statement executed, listing timing information per call rather than the total per subprogram. Because this option can generate a large volume of output, source code directives FLOW and NOFLOW should be used to include only areas of special interest.

SETPLIMQ is requested by a subroutine call in the source code. In Fortran this call would appear as follows.

```
      CALL SETPLIMQ(count)
```

In general, the above statement should be one of the first executable statements in a program. The value of *count* specifies the number of trace lines printed. Since one line is produced for each CALL and each RETURN, *count* should be set to twice the number of CALL statements for which flowtracing is desired.

In effect, each CALL and each RETURN statement is given a sequence number at run time; if this number is less than ABS(*count*) for a given CALL or RETURN, a line is printed to $OUT; these lines are interspersed with other output. CALL or RETURN statements executed before CALL SETPLIMQ are counted toward the line limit but generate no output.

SETPLIMQ output example:

```
FLOWTRACE  --  Call   SOLVDE   by SFROID      407727978   407301664
FLOWTRACE  --  Call   DIFEQ    by SOLVDE      407744474   407743808
FLOWTRACE  --  Return DIFEQ    to SOLVDE            149         149
FLOWTRACE  --  Call   PINVS    by SOLVDE      407775007   407743808
FLOWTRACE  --  Return PINVS    to SOLVDE            844         844
FLOWTRACE  --  Call   DIFEQ    by SOLVDE      407806315   407743808
FLOWTRACE  --  Return DIFEQ    to SOLVDE            382         531
```

The longer numbers such as 407727978 are reference times shown in CPU clock periods. Shorter numbers such as 149 indicate how many clock periods were used by each call. The shorter numbers in the left column indicate how many clock periods were actually charged to a subprogram; these numbers can be used to assess program performance. Shorter numbers in the right column include FLOWTRACE overhead.

SPY - TIMING SELECTED CODE SEGMENTS 4

SPY indicates how much time is being spent in different segments of code within routines. Routines measured by SPY can be written in any language available on Cray computers running COS. An advantage of SPY is that it entails almost no overhead. (A *routine* here is any identifiable, executable code with distinct entry and end points, whether it is part of your program or contained in a library.)

When so requested by SPY, COS reads the address of the instruction currently being executed. Addresses are grouped in *buckets*, whose size is selectable; each access of a bucket is called a *hit*. After execution, SPY outputs statistics for all the buckets, including a bar graph on which "spikes" indicate code where the most time is spent.

Because of the time interval between samples, SPY's results involve probability: in the worst case, an address could be executed many times but never be sampled by SPY. The TS and BS parameters on the SPY control statement change the address and time intervals to provide a check on the probability.

SPY's output includes a line for each bucket that was hit more than a specified number of times, followed by totals organized by subroutine. Because the volume of this output can be considerable, the SUB keyword is provided to let you specify which routines are to be represented.

SPY results contain an inherent error which is insignificant for most applications. However, if your application program is very short, the following sources of noise may be greater than the real information gathered:

- Loading CSP to crack your program's control statement
- Loading your program
- Loading CSP to crack the control statement which specifies the SPY, POST action
- Loading SPY, turning off the COS SPY activity and obtaining the raw data

The execution time for most application programs is long enough to make these noise sources insignificant.

The following examples show how SPY is used in three typical situations. In all three cases, the program being spied is named SUBJECT.

CFT77 with SEGLDR

```
CFT77,DEBUG.
SEGLDR,CMD='ABS=SUBJECT'.     * Loader does not run program
SPY,PREP.
SUBJECT.                      * Program executes
SPY,POST.
```

CFT with LDR

```
CFT,ON=IZ.
LDR,NX,AB=SUBJECT.            * Loader does not run program
SPY,PREP.
SUBJECT.                      * Program executes
SPY,POST.
```

CAL with SEGLDR

```
CAL,SYM.
SEGLDR,CMD='ABS=SUBJECT'.
SPY,PREP.
SUBJECT.
SPY,POST.
```

4.1 REQUIREMENTS FOR RUNNING SPY

In addition to normal compiling, loading, and executing of the subject program, the JCL needed for using SPY must accomplish the following:

- Generate a debug symbol table for the subject program
- Load and execute the subject program in two separate steps
- Execute SPY twice: both before and after execution of the subject program

4.1.1 GENERATING THE SYMBOL TABLE

SPY requires a symbol table for the subject program, generated by the appropriate language processor. The table is sent to dataset $BLD by default, from which it is copied by the loader into $DEBUG. The table is generated by different processors using the following control statement specifications:

Processor	Keyword
CFT77	DEBUG
CFT	ON=IZ or DEBUG (see below)
CC	-Z (deferred implementation)
PASCAL	O=DM3
CAL	SYM

Note that the DEBUG option on CFT turns off all optimization. This makes the result inapplicable for many situations.

4.1.2 LOADING AND RUNNING THE SUBJECT PROGRAM

When SPY is used, the subject program is loaded and run in separate steps. Between the load and run steps, SPY,PREP is run (see subsection 4.1.3). The Cray loaders SEGLDR and LDR perform these separate steps as follows for program SUBJECT:

```
SEGLDR,CMD='ABS=SUBJECT'.               LDR,NX,AB=SUBJECT.
...                                     ...
SUBJECT.                                SUBJECT.
```

4.1.3 THE PREP AND POST RUNS

To use SPY you must include the SPY command both before and after the command line for the subject program, with the keywords PREP and POST, respectively.

The SPY,PREP call reads the symbol table and calculates the absolute address of each symbol. Then, as specified by the TS, BS, and THRSHLD parameters, SPY performs one or more calls to COS, specifying regions of interest. Each call sets up a table so that at each interupt, COS reads the P register in your exchange package and increments the bucket that includes that P address. This information is written to dataset $SPY. A SPY,POST call then reads $SPY and generates a report as specified by the SUB, THRSHLD, and NOLIB parameters.

4.2 SPY CONTROL STATEMENT

The format for the PREP call of SPY is as follows:

SPY,PREP [,BS=*bcktsz*] [,D=*dbugdn*] [,S=*scrtch*]

[,SUB=rtn_1:rtn_2:...rtn_n] [,THRSHLD=*gap*] [,TS=*time*].

BS=*bcktsz*	Bucket size in words; default, 4. Each bucket begins on a word address that is a multiple of the bucket size. This parameter affects storage requirements; see subsection 4.3.
D=*dbugdn*	Dataset containing the symbol table for the subject program. The default is $DEBUG, which is the dataset to which SEGLDR and LDR write the symbol table (after copying it from $BLD).
S=*scrtch*	Dataset where the tables prepared by SPY,PREP are to be written, and from which SPY,POST is to read these tables. Default is $SPY.
SUB=rtn_1:rtn_2:...rtn_n	List of routines that are to be reported on; $n \leq 20$. The default is to report on the entire executable program; that is, the SUB= parameter is used to limit output. This parameter affects storage requirements; see subsection 4.3.
THRSHLD=*gap*	This keyword is not needed when SEGLDR is used. It is used to make the best use of storage when LDR is used; see subsection 4.3.
TS=*time*	Time slice in microseconds; default, 500.

The format for the POST call of SPY is as follows:

SPY,POST [,ADDRESS] [,L=*listdn*] [,NOLABEL] [,NOLIB]

[,S=*scrtch*] [,SUB=rtn_1:rtn_2:...rtn_n] [,MINHIT=*n*].

ADDRESS	Causes a report by address rather than by label
L=*listdn*	Dataset on which the final report is written; default, $OUT.

NOLABEL	Causes the bucket size to be an entire routine. This makes the output similar to that of FLOWTRACE, except that time spent in library calls is accounted for separately.
NOLIB	Excludes library calls by eliminating any reference to a routine name that begins with a dollar sign ($). Library calls that do not begin with $, such as SECOND, are not affected by the NOLIB keyword.
S=*scrtch*	Dataset where the tables prepared by SPY,PREP are to be written, and from which SPY,POST is to read these tables. Default is $SPY.
MINHIT=*n*	Minimum number of hits required for an output line to be printed for a bucket or label; default 1. (MINHIT=0 is not recommended.)
SUB=rtn_1:rtn_2:...rtn_n	List of routines that are to be reported on; $n \leq 20$. The default is to report on the entire executable program; the SUB= parameter is used to limit output. This parameter affects storage requirements; see subsection 4.3.

4.3 STORAGE REQUIREMENTS

Using SPY causes COS to allocate storage in the Job Table Area (JTA) of your job. This space counts against the maximum memory allowed to your job. Smaller bucket sizes require more JTA memory than larger sizes.

Several factors influence the amount of JTA storage required by a given symbol table:

BS	One JTA word is required for each bucket monitored in the user field. Larger bucket sizes provide less resolution but require less JTA space. The default is four words.
THRSHLD=*t*	When you use LDR, areas of memory for code are intermingled with those for data. Since spying in data areas serves no purpose, the system jumps past any data area as large as *t* words (default 512), to the next code area. This saves the JTA space that would be used for buckets in the data area, but requires more JTA space for a separate call to SPY. THRSHLD allows you to adjust *t* for individual cases, in order to use the least JTA space. If this is a significant factor in your program, you should consider switching to SEGLDR, which gathers all code areas together.

SUB The SUB parameter can be used to limit SPY activity to 20 or fewer subprograms. Provided that using such a subset gives you the information you need, you can greatly reduce the JTA requirements this way.

The default choices for the preceding parameters almost always yield a successful run, so they seldom need adjustment. However, if you encounter memory limitations, adjusting these parameters should allow you to get the information you need.

4.4 SPY OUTPUT

The main SPY output shows seven columns, giving the following information:

Sample SPY output:

```
ROUTINE  LABEL     ADDRESS   HITS  %PRG  %SUB   SECONDS  CUM%
                                                               |
MAIN     MAIN      00000201    12   7.6 100.0     0.006   7.6  |*******
                                                               |
SUB1     110A      00001400    13   8.3  44.8     0.006  15.9  |********
SUB1     306       00001512     6   3.8  20.7     0.003  19.7  |***
SUB1     00003     00001541     3   1.9  10.3     0.001  21.7  |*
SUB1     115       00001604     3   1.9  10.3     0.001  23.6  |*
SUB1     110D      00001615     1   0.6   3.4     0.000  24.2  |
SUB1     110B      00001620     3   1.9  10.3     0.001  26.1  |*
                                                               |
SUBZZZ   SUBZZZ    00003177     4   2.5  80.0     0.002  28.7  |**
SUBZZZ   40        00003240     1   0.6  20.0     0.000  29.3  |
                                                               |
$CDCO    $CDCO     00004140     4   2.5 100.0     0.002  31.8  |**
                                                               |
$IBMO    $IBMO     00004561     1   0.6 100.0     0.000  32.5  |
                                                               |
$RLD     $RLD      00007104     1   0.6 100.0     0.000  33.1  |
                                                               |
$WLD     $WLD      00013162     2   1.3 100.0     0.001  34.4  |*
                                                               |
$NICV    $NICV     00021640     6   3.8 100.0     0.003  38.2  |***
                                                               |
$NOCV    $NOCV     00022217     5   3.2 100.0     0.002  41.4  |***
                                                               |
$STOP    $STOP     00022700     4   2.5 100.0     0.002  43.9  |**
                                                               |
$GTDSP   $GTDSP    00024321    10   6.4 100.0     0.005  50.3  |******
                                                               |
$RCW     $RCW      00025100     2   1.3 100.0     0.001  51.6  |*
                                                               |
$WCH     $WCH      00034452     3   1.9 100.0     0.001  53.5  |*
                                                               |
```

Header	Meaning
ROUTINE	Name of the routine using this address. Note that routine names beginning with $ are library routines and are documented in the Programmer's Library Reference Manual and the System Library Reference Manual, CRI publications SR-0113 and SM-0114. Many of these names suggest their functions, mathematical or otherwise. Routines whose names begin with $W and $R are often for writing and reading, respectively.
LABEL	The label used by SPY to identify the bucket. The label can be any of the following: - The name of the program unit. This label is given to the first bucket in each subprogram. - An integer followed by an alphabetic character. The integer is the label for a DO loop, and the character is used to identify more than one bucket in a single DO loop; for example, 11B is the label for the second bucket for the DO 11 loop in a program unit. - A five-digit integer. SPY generates these labels for buckets to which neither of the above cases applies.
ADDRESS	Octal address of the bucket's first word
HITS	Number of hits for this bucket
%PGM	Percent of hits in program: hits for this label divided by total hits in the program, multiplied by 100
%SUB	Percent of hits in subroutine: hits for this label divided by total hits in subroutine
SECONDS	Estimated time: number of hits multiplied by time slice
CUM%	Cumulative sum of %PRG column.
Far right	Bar graph showing column 5, %PRG, percentage of hits in the program

Sample:

```
ROUTINE SUMMARY     HITS   %PRG     SECONDS

MAIN                  12    7.6       0.006      |%%%%%%%
SUB1                  29   18.5       0.014      |%%%%%%%%%%%%%%%%%%
SUBZZZ                 5    3.2       0.002      |%%%
$CDCO                  4    2.5       0.002      |%%
$IBMO                  1    0.6       0.000      |
$RLD                   1    0.6       0.000      |
$WLD                   2    1.3       0.001      |%
$NICV                  6    3.8       0.003      |%%%
$NOCV                  5    3.2       0.002      |%%%
$STOP                  4    2.5       0.002      |%%
$GTDSP                10    6.4       0.005      |%%%%%%
$RCW                   2    1.3       0.001      |%
$ATAN2                 7    4.5       0.003      |%%%%
$CABS                 27   17.2       0.013      |%%%%%%%%%%%%%%%%%
$DASS                  1    0.6       0.000      |
                                                 |
UNDER                  0    0.0       0.000      |
BETWEEN                0    0.0       0.000      |
OVER                  20   12.7       0.010      |%%%%%%%%%%%%
RUN SUMMARY          157              0.078
```

After the information for all the buckets, a summary by routine is printed. The following columns are used:

HITS	Total hits for the routine
%PRG	Percent of hits in program: hits for this routine divided by total hits in program, multiplied by 100
SECONDS	Estimated time in seconds: number of hits multiplied by time slice
Far right	Bar graph showing column 3, %PGM, percentage of hits in the program

At the bottom of the routine summary are the categories UNDER, BETWEEN, OVER, and RUN SUMMARY. The first three refer, respectively, to addresses that are lower than any area that was supposed to be measured, between such areas, or higher than these areas. If the UNDER and BETWEEN counts are significantly more than zero when the entire program is being measured, it indicates improper use of SPY. A high number for OVER can have various causes that do not indicate a problem, such as the use of overlays. The run summary is a total of all routines, including these final special categories.

The estimated time should be similar to that generated by FLOWTRACE, with the qualification that FLOWTRACE includes time spent by library calls in the timing of the calling routine.

PERFMON - CRAY X-MP PERFORMANCE MONITOR

PERFMON is an interface to the hardware performance monitor available on CRAY X-MP series computers. It is invoked within JCL.

To use PERFMON, first execute it to turn on one of four available performance monitors. Then execute the program whose performance is to be monitored. To obtain the PERFMON report, execute PERFMON with the RESET or REPORT option immediately after the subject program.

PERFMON results can be affected by the times taken for loading your program and loading CSP to crack your program's control statement. These are a factor only for very short applications.

5.1 PERFMON CONTROL STATEMENT

The format of the PERFMON control statement is as follows:

```
PERFMON [,L=list] ,result.
```

L=*listdn*	Specifies the listing output dataset for PERFMON reports. The default is $OUT.
result	Specifies the primary effect of the PERFMON control statement; can be one of the following:

ON=*n*	Turns on performance monitoring and selects one of four hardware performance monitors. These are indicated in PERFMON output at the upper right in each block as "Monitor 0" and so on. The monitors record the following:

0	Execution summary
1	Hold issue conditions
2	Memory activity
3	Vector events and instruction summary

OFF	Turns off performance monitoring

REPORT	Produces a report of the current performance monitor counters (to the L listing file). It does not clear the counters.
RESET	Produces a report of the current performance monitor counters (to the L listing file). In addition, the counters are cleared after printing and monitoring is turned off.

JCL example for program SUBJECT:

```
SEGLDR,CMD='ABS=SUBJECT'.
PERFMON,ON=0.
SUBJECT.
PERFMON,RESET.
PERFMON,ON=1.
SUBJECT.
PERFMON,RESET.
PERFMON,ON=2.
SUBJECT.
PERFMON,RESET.
PERFMON,ON=3.
SUBJECT.
PERFMON,RESET.
```

5.2 PERFMON OUTPUT

Each run of PERFMON (that is, each monitor used) generates one block of information.

Output sample of monitor 0, execution summary:

```
Cycles: 6173669, CP seconds:  0.058649856, Task number: 1, Monitor 0.

     36.95 Million instr/sec (MIPS)         2166976 instructions issued
      2.85 avg. clock periods per instruction.
     43.87 % of CP holding issue.           2708154 CP holding issue
 407861.19 instr. buf. fetches/sec.           23921 fetches
      0.00 I/O memory references/sec.             0 I/O references
6528183.18 CPU memory references/sec.        382877 CPU references
2972522.24 Floating adds/sec.                174338 F.P. adds
4045108.01 Floating multiplies/sec.          237245 F.P. multiplies
 710419.48 Floating reciprocals/sec.          41666 F.P. reciprocals
      7.73 Million Floating Operations/sec (MFLOPS)
```

Output sample of monitor 1, hold issue conditions:

```
Cycles:  6302128, CP seconds:    0.059870216, Task number:   1, Monitor 1.

 0.00 % CP hold issue - semaphores
 0.01 % CP hold issue - shared registers
 6.26 % CP hold issue - A-regs & A-functionals
36.13 % CP hold issue - S-regs & S-functionals
 0.16 % CP hold issue - V-regs
 0.24 % CP hold issue - V-functionals
 0.47 % CP hold issue - scalar memory
 0.74 % CP hold issue - block memory
```

Output sample of monitor 2, memory activity:

```
Cycles:  6387292, CP seconds:  0.060679274, Task number:  1, Monitor 2.

 412694.46 instr. buffer fetches/sec.
4413154.32 scalar memory refs/sec.
     40.94 % conflicts/scalar reference
  33751.23 Block memory refs/sec.
     19.63 % conflicts/block reference
2050156.37 Block memory refs/sec.
     18.51 % conflicts/block reference
1303344.53 vector memory references/sec.
```

Output sample of monitor 3, vector events and instruction summary:

```
Cycles:  5069820, CP seconds:  0.048163290, Task number:  1, Monitor 3.

  4410496.04 000-017 instructions/sec issued,
           12.03 % of all instructions
 32123345.39 scalar (020-137) instructions/sec issued,
           87.64 % of all instructions
    14554.65 vector integer&logical instr./sec. issued,
            0.04 % of all instructions
    37123.71 vector floating-point instr./sec. issued,
            0.10 % of all instructions
    66772.85 vector load-store instr./sec. issued,
            0.18 % of all instructions
   412409.53 vector integer operations/sec.   28.34 elements/instruction
   917337.67 vector floating operations/sec.  24.71 elements/instruction
  1555811.49 vector memory references/sec.    23.30 elements/instruction
```

INDEX

INDEX

READER'S COMMENT FORM

COS Performance Utilities Reference Manual SR-0146

Your reactions to this manual will help us provide you with better documentation. Please take a moment to check the spaces below, and use the blank space for additional comments.

1) Your experience with computers: ____ 0-1 year ____1-5 years ____5+ years
2) Your experience with Cray computer systems: ____0-1 year ____ 1-5 years ____5+ years
3) Your occupation: ____ computer programmer ____ non-computer professional
   ____ other (please specify): ____________________________
4) How you used this manual: ____ in a class ____as a tutorial or introduction ____ as a reference guide
   ____for troubleshooting

Using a scale from 1 (poor) to 10 (excellent), please rate this manual on the following criteria:

5) Accuracy _____	8) Physical qualities (binding, printing) ____
6) Completeness ____	9) Readability ____
7) Organization ____	10) Amount and quality of examples ____

Please use the space below, and an additional sheet if necessary, for your other comments about this manual. If you have discovered any inaccuracies or omissions, please give us the page number on which the problem occurred. We promise a quick reply to your comments and questions.

Name ______________________	Address ____________________
Title ______________________	City ______________________
Company ___________________	State/ Country _______________
Telephone __________________	Zip Code ___________________
Today's Date ________________	

CUT ALONG THIS LINE

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY
RESEARCH, INC.

Attention: PUBLICATIONS
1345 Northland Drive
Mendota Heights, MN 55120

FOLD

STAPLE

READER'S COMMENT FORM

COS Performance Utilities Reference Manual SR-0146

Your reactions to this manual will help us provide you with better documentation. Please take a moment to check the spaces below, and use the blank space for additional comments.

1) Your experience with computers: ____ 0-1 year ____1-5 years ____5+ years
2) Your experience with Cray computer systems: ____0-1 year ____ 1-5 years ____5+ years
3) Your occupation: ____ computer programmer ____ non-computer professional
   ____ other (please specify): ____________________
4) How you used this manual: ____ in a class ____as a tutorial or introduction ____ as a reference guide
   ____for troubleshooting

Using a scale from 1 (poor) to 10 (excellent), please rate this manual on the following criteria:

5) Accuracy _____
6) Completeness ____
7) Organization ____
8) Physical qualities (binding, printing) ____
9) Readability ____
10) Amount and quality of examples ____

Please use the space below, and an additional sheet if necessary, for your other comments about this manual. If you have discovered any inaccuracies or omissions, please give us the page number on which the problem occurred. We promise a quick reply to your comments and questions.

Name ____________________
Title ____________________
Company ____________________
Telephone ____________________
Today's Date ____________________

Address ____________________
City ____________________
State/ Country ____________________
Zip Code ____________________

1026415A4

CUT ALONG THIS LINE

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY
RESEARCH, INC.

Attention: PUBLICATIONS
1345 Northland Drive
Mendota Heights, MN 55120

FOLD

STAPLE