Department of Applied Analysis
and Computer Science
Technical Report CSTR 1010
August 1971

FORTSTAT - A FORTRAN TIMING PROGRAM

by
W. Morven Gentleman
L. Bryan Douglas

GENERAL DESCRIPTION

This program is designed to run an executable load module produced by either the FORTRAN G or H language translators and produce statistics about the number of times certain routines are called and the total amount of time spent in those routines. More specifically, the load module must contain routines which were produced by the FORTRAN translators, although not all the routines in the module need be so produced; in fact, the only one which must be FORTRAN-produced is the mainline program. Only FORTRAN-compatible routines will be timed so these will usually be in the majority.

We may not always wish to time all the routines in our program but may instead want to group some routines in with the calling routines instead of gathering statistics on them separately. For example, we may have a small utility subprogramme which is called from many routines in the program and we may not care how much time is spent in that routine itself but would rather see this time included with whatever called it. It is possible to specify which routines are to be timed; all others will be ignored and treated as if they were part of the calling routine. If one of these routines which is not being timed calls a routine which is being timed (call it GEORGE) then it is treated just as if GEORGE were called from the next-higher-level

timed routine. Figure 1 illustrates this concept.

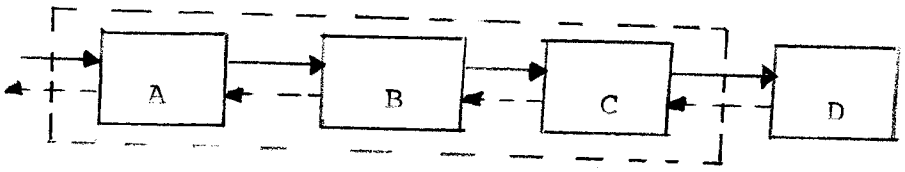


Fig. 1

As far as the timing routine is concerned, everything in the dashed box is routine A.

Here routine A calls B which calls C which calls D. A and D are being timed and B and C are not. Because the timing routine does not know that B and C are there, it ignores them completely and treats them as part of A. Thus, when A is called we start counting its time, it calls B which calls C and this time is still credited to A, then C calls D and we start the timing for a new routine (D), the time now being counted to D until it returns to C at which point the time gets credited to A again.(breathe) The time continues to be credited to A until we get back to where it returns. Any Assembler or other language routines will be ignored in this same manner unless they conform to FORTRAN linkage and prologue conventions. These conventions will be explained later.

The output of the program consists of a listing of all the timed routines with the number of times the routine was called and the total time spent in the routine, in seconds. Totals for the entire run are also given. The

mainline program will be marked with an asterisk and will be timed whether it is explicitly told to be or not. There is an overhead of approximately 500 microseconds per CALL to a routine being timed; this is unavoidable but must be taken into consideration when looking at the results. This means that about .01 seconds must be subtracted from the given time for every 20 calls indicated. This time is only approximate and can vary considerably either way, but over a large number of calls it should be fairly close.

Times given may vary considerably from run to run but should be reasonably consistent within a given run. This is due to the fact that OS/360 does not give completely accurate times. Therefore times should be used only for comparisons relative to other routines in the same run.

Core required by FORTSTAT is approximately 4K. The exact amount will depend on how many routines are being timed and whether or not the option of specifying the routines to be timed is used.

USER INSTRUCTIONS

This program has been written to be as easy to use as possible. It is not necessary to do anything to the translator, to make any special runs, or to modify the program to be timed. All that is needed is an executable load module with FORTRAN G or H routines in it. The program will scan through the load module and pick out all those routines which it recognizes as FORTRAN-compatible. The only JCL that is needed is the following.

```
//  EXEC PGM=FORTSTAT,PARM='progname'
//MODULE DD DSN=library,etc.
//STATOUT DD SYSOUT=A
```

Here "library" is the name of the data set containing the load module, "progname" is the name of the member within "library" and "etc." is anything else you may need depending on whether the data set is catalogued etc. STATOUT is the DD card for the output statistics (it is BLKSIZE=133,DSORG=PS,RECFM=FBA).

The following changes must be made to the JCL if you wish to explicitly specify which routines are to be timed.

```
// EXEC PGM=FORTSTAT,PARM='progname,'
//STATIN  DD ????
```

The comma after "progname" in the PARM field tells the program to look for STATIN and to read from it the names

of the routines to be timed. The MODULE and STATOUT DD cards must still be included but you must also add the card for STATIN. "????" will usually be * indicating the input stream but may be any data set of 80-byte fixed records. The names of the routines to be timed are placed one per card anywhere in the first 72 columns. All imbedded blanks are removed, so the first 8 non-blank characters on the card will be used as the name (if there are less than 8 characters the name will be padded on the right with blanks). Thus, as the program scans through the load module, only the mainline program and those routines which match a name on one of the cards will be timed. No check is made to see if there are names on the cards which do not match routines in the module.

PROGRAM LOGIC

This program has the option of timing all the routines in the module or having specified explicitly which routines are to be timed. The name of the module to be timed is passed in the PARM field at execution time and if there is a comma following the name then we know that the names of the routines to be timed will be specified on DDname STATIN. The name from the PARM field is moved into a BLDL list to be used to FIND the module; if there is a comma, we set NAMECNT to 1 so we will know later to read STATIN, otherwise it is set to 0. We then issue a BLDL macro with the name we got from the PARM field. In this way we only search the directory once and we can use the resulting information for both a fetch and a read. If the BLDL was not accomplished properly we print a message indicating either that the module was not found in the library or that there was an I/O error in reading the directory and in either case we terminate execution at this point.

If the BLDL went okay then we load the appropriate module, do a FIND to set the read pointer to the beginning of the module, and then issue a conditional GETMAIN for between 0 and 16,000 bytes. The GETMAIN will get all the core available, up to 16,000 bytes, so we must do the GETMAIN after the LOAD or there will be no core to LOAD the module into.

Now NAMECNT is checked and if non-zero we open the DCB for STATIN and start reading the names of the routines to be timed. We now set up a table at the bottom of the GETMAINed core area. Each entry is 8 bytes and contains one of the names specified on STATIN. When this table is complete we store the address of the end of the table in GADDR so we will know where we can start building the statistics-table.

Here we come to the part where we scan through the loaded program and find all those routines which look like they might be FORTRAN. When we find a FORTRAN routine we set up a table entry in the following format:

```
TNAME: 8 bytes  -  this is the module name
TADDR: 1 word   -  entry point of the routine
TCALLS: 1 word  -  number of times the routine was called
TTIME: 1 word   -  time spent in that routine
TRET:  1 word   -  return address from the routine
                   the last time it was called
TPREV: 1 word   -  pointer to entry for calling routine
```

When the table is set up the name and address are inserted and all other fields are set to 0. The DSECT in the program which is used to access this table is called TABLE.

Here is how we scan to find the Fortran routines; from the BLDL we know the relative entry point of the routine and from the LOAD we know the absolute entry point,

we subtract to find the absolute load point of the module and use this as a base for all the relative addresses we get from the ESD. We now start reading the load module and read the CESD records which are at the beginning of the module. From this we find out the name and relative address of every CSECT or ENTRY point and we now can calculate the real address. Once we know the address we look to see if this looks like a FORTRAN routine. The first thing we look at is the first word in the routine and this should be X'47F0F00C', if not we throw that away and try the next one. If the first word matches okay then we look at the next 8 bytes and get out the name, the first of these 8 bytes will be X'06' for Fortran G and X'07' for Fortran H. The name will be in the next 7 bytes and is in a different format depending on whether it is G or H. For FORTRAN G the name is left justified and padded on the right with blanks, for FORTRAN H it is right justified and padded on the left with X'00'. We get the name out and check it against the name in the ESD. If the names match then we look to see if there is a table of names to be timed (ie. NAMECNT≠0) and if so we look the name up in the name-table (linear search).

If the name is in the name-table or there is no table then we set up a statistics-table entry for that name. As we are doing this we check each address to see if it matches the entry point address of the module for if it does

it is the mainline routine and is timed whether specified or not. All the time we are scanning through the ESD information we are looking at the names for IBCOM# and when we find this we save the entry address becuase we will need it later. At the same time that we are setting up the table entries we are also changing the entry to the routine. The first word was X'47F0F00C' and we change this so that the first byte is X'00' and the next 3 bytes point to the table entry for that routine. When we have set up the entire table, the number of entries in the table is saved in TABCOUNT. At this point, any unused core is FREEMAINed.

Since we changed the first byte in each routine to a X'00' we are going to get an operation exception as soon as we enter the routine. We must therefore change the FORTRAN SPIE exit so that we will get control in case of a program interrupt. Remember that as we were reading through the ESD information we saved the address of IBCOM#. We now start at the entry point address of the loaded module and search word by word until we find this address. which is the adcon that the FORTRAN mainline will use to branch to IBCOM# to set up the SPIE, so we save this address and replace it with the address of our own routine. FORTRAN branches 64 bytes past the entry point of IBCOM# to do the initialization, so we actually put in the address 64 bytes before the routine we want control passed to.

At this point we are ready to start executing the FORTRAN program. We set a pointer saying that the current routine is the mainline, put an appropriate return address in the table entry for the mainline, set the timer going and branch to the mainline program. The first thing the FORTRAN program does is branch to IBCOM# to set up the error routines; since we fudged its adcon, it comes to us instead. This is too early for us to do anything, so all we do is change register 14 so that when IBCOM# returns it will come back to us. We then branch to IBCOM# as FORTRAN wanted to. When IBCOM# returns to us we go in and change the exit address in the PICA so that a program interrupt will transfer control to us instead of to the FORTRAN error routine. We save the address of the FORTRAN error routine since we will pass to it any interrupts which we do not want. At this time we also set the bit in the PICA to indicate that it is enabled for operation exceptions and we return to the FORTRAN mainline.

The FORTRAN program is now executing semi-normally and we are ready to time the routines. I say semi-normally because everything is normal except for the entry to, and exit from, the timed routines. When control is passed to one of the routines which we are timing we get an operation exception and control is passed to our exit routine; we check the interrupt to see that it was one of those that we

generated and if not we pass it to the FORTRAN error routines. If it passes this test then we are ready to do the timing; we first get the previous clock time and a pointer to the calling routine (CLOCK and CURRTN). We issue a TTIMER to get the current time and subtract this from the previous time (the clock is counting down), we add this time to the total time for the previous routine and set CURRTN to point to the new routine. We put the new clock time in CLOCK and add 1 to the number of calls for the new routine. We also save (in the table entry for the new routine) the pointer to the calling routine (so we can get back to it when we return from this routine) and the real return address from this routine. We now change register 14 so that the routine will return to our other timing routine and we branch to the new routine.

When we return from the routine we do exactly the opposite of what we did on entering. We do a TTIMER to find the time spent in the routine and add it to the total (also put the new time in CLOCK). We get the pointer to the previous routine from the table and put it in CURRTN, get the return address from the table and go there.

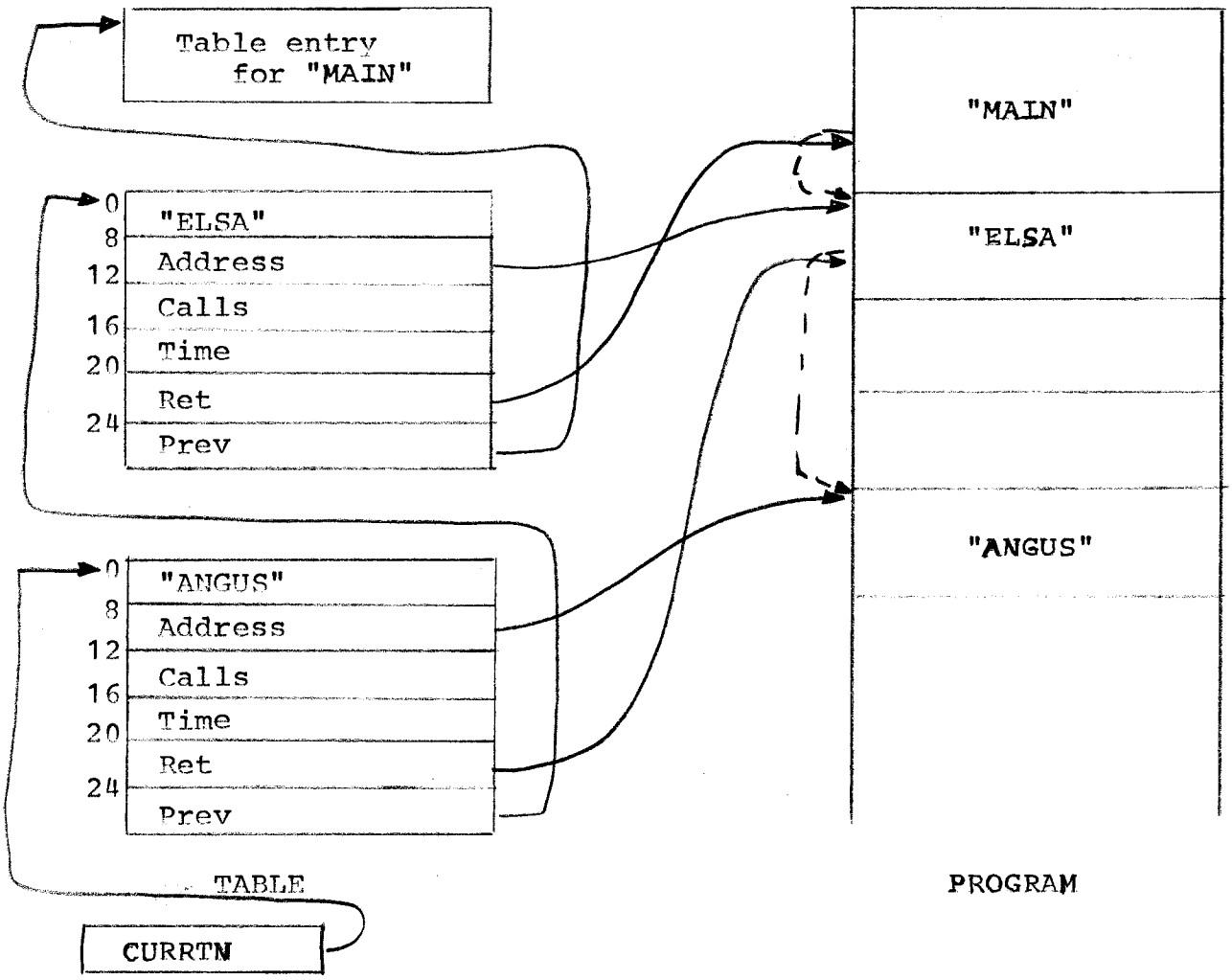
When we started executing the mainline program we put in the return address portion of its table entry, the address of the place we want it to return in the control program; therefore the timing of the mainline is handled

exactly the same as every other routine and we need do nothing special at the end. When control is passed back to us when the FORTRAN program is all finished, we merely scan through the table , print out all the statistics and we are finished.

NOTE:

Appendix A contains a diagram illustrating run-time pointer usage and Appendix B contains a sample of output.

APPENDIX A



RUN-TIME POINTER STRUCTURE

"ELSA" was called from "MAIN" and in turn called "ANGUS" which is currently executing. RET in the table entry for "ANGUS" is the actual return address (R14 contains the address of the timing routine.) The dotted line shows the actual calls in the program being timed.

APPENDIX B

This Appendix contains a sample output from FORTSTAT showing the format of this listing.

ROUTINE	CALLED	TIME(SEC)
*MAIN	0	.27
SETCM1	1	.00
USHER	1	.00
CHCP	558	.80
PAGER	140	.17
SETSYM	5	.00
FABRIC	5	.02
DIVEST	139	.10
GOSSIP	1	.03
GETLAB	2	.01
GETNUM	0	.00
TINKER	0	.00
REWIND	2	.09
INCARD	137	.49
IOID	1	.14
IDFBIN	1	.01
SIMILE	344	.29
MOVE	267	.20
IMOV	15	.01
LOCK	106	.32
MATCH	941	.56
IOTAPE	15	.07
MEMBER	0	.00
ADAM	0	.00
PRNCIT	14	.62
CARCIT	15	.50
SETCOM	1	.00
SCOUT	650	.60
SYMBOL	39	.18
GETCH	405	.41
JUST	210	.40
SCAN	0	.00
ARTY	105	.19
IDEAL	405	.25
TOTAL	4525	6.87

APPENDIX C

This Appendix contains a listing of the FORTHCLG catalogued procedure used at the University of Waterloo and a copy of the users guide for FORTSTAT which was written for use at the University of Waterloo. The JCL given in this guide requires that the catalogued procedure agree with that given in the relevant aspects.

FORTSTAT - A FORTRAN TIMING PROGRAM

USER'S GUIDE

FORTSTAT will run an executable FORTRAN G or H load module and will compile statistics on the number of times each subroutine or each entry point is called and also on the amount of time spent in each routine. If there are separate entry points within a subroutine, each entry point is treated as a separate routine.

The program has been written so that it will be extremely easy to use. It is not necessary to do anything to the translator or to change the program being timed. All that is required is an executable load module with FORTRAN G or H routines in it.

The JCL necessary for running a FORTRAN G compile,linkedit and go with FORTSTAT is the following:

```
// EXEC FORTGCLG,LMOD=FORTSTAT,GO=FORTGCLG
//GO.DELETE DD DUMMY
//GO.MODULE  DD  DSN=&&GOSET,DISP=(OLD,DELETE)
//GO.STATOUT   DD  SYSOUT=A
```

For FORTRAN H use FORTHCLG on the EXEC card, both as the procedure name and in the GO= parameter field.

If the user wishes to specify the routines to be timed, the EXEC card must be changed to

```
// EXEC FORTGCLG,LMOD=FORTSTAT,GO='FORTGCLG,'
```

and he must include

```
//GO.STATIN
```

which points to a data set containing the names of the routines to be timed. The format of this data set will be explained later in this document.

NOTE: The GO.DELETE card must be the first DD card for the GO step while the GO.MODULE and GO.STATOUT cards go after the cards which overide the cards in the procedure, i.e. GO.FT05F001, GO.SYSIN etc.

EXAMPLE: Suppose we have a FORTRAN program which will be read from SYSIN and which contains the routines A,B,C,D and E. If we wish to time all the routines we would use the following procedure:

```
// job
// EXEC FORTGCLG,LMOD=FORTSTAT,GO=FORTGCLG
//FORT.SYSIN  DD *
    program
/*
//GO.DELETE DD DUMMY
//GO.SYSIN  DD *
    data
/*
//GO.MODULE  DD  DSN=&&GOSET,DISP=(OLD,DELETE)
//GO.STATOUT  DD  SYSOUT=A
```

Now if we wish to time only routines B and C we would change GO=FORTGCLG to GO='FORTGCLG,' in the EXEC card and add after //STATOUT the following:

```
//GO.STATIN  DD *
B
C
/*
```

The necessary JCL to run FORTSTAT if the load module is already available is as follows:

```
// EXEC PGM=FORTSTAT  PARM='progname'
//MODULE  DD  DSN=library,etc.
//STATOUT  DD  SYSOUT=A
```

'Library' is the name of the data set containing the load module; 'progname' is the name of the member within 'library' and 'etc' is anything else the user may need depending on whether or not the data set is catalogued.

STATOUT is the DD card for the output statistics; it is BLKSIZE=133,DSORG=PS,RECFM=FBA. FORTSTAT will time all routines included in the load module unless otherwise specified as follows:

```
// EXEC PGM=FORTSTAT,PARM='progname,'
//STATIN DD  ????
```

The comma following 'progname' in the PARM field indicates to the program that STATIN is following and so it will read from it the names of the routines the user wishes to time. The user must still include the MODULE and STATOUT cards,but must also include the STATIN card. '????' will generally be * indicating the input stream but it may be any data set of 80 byte fixed records.

The names of the routines to be timed are placed one per card anywhere in the first 72 columns. All embedded blanks are removed so the first 8 non-blank characters on the card will be used as the name(of course if there are

less than 8 characters they will be padded on the right with blanks). Thus, as the program scans through the load module, only the mainline program and those routines which match a name on one of the cards will be timed. No check is made to see if there are names on the cards which do not match routines in the module.

The output statistics list the routine, the number of times that the routine was called and the total time spent in the routine in seconds.

With each call to a subroutine there is an overhead of approximately .5ms/call. This overhead is included in the total time spent and so must be subtracted from this value to arrive at the actual amount of time spent in the particular routine.

Note that these should be used as relative times not absolute as they may vary from run to run.

Core required is approximately 4K more than normal for the timed program.

FORTGCLG Catalogued Procedure

```
//FORTGCLG PROC  PROG=IEYFORT,LIB='SYS1.FORTLIB',CORE=96K,FSIZE=128K,
//         FORT=,LINK=LINKEDIT,LKED=LIST,LSIZE=96K,EL=4,        07SEP71
//         PROGRAM=FORTGCLG,LMOD='*.LKED.SYSLMOD',GO=,LINKERR=4
//FORT     EXEC  PGM=&PROG,REGION=&FSIZE,PARM='&FORT'
//STEPLIB  DD  DSNAME=SYS1.R19.LINKLIB,DISP=SHR
//SYSPRINT DD  SYSOUT=A
//SYSPUNCH DD  SYSOUT=B
//SYSLIN   DD  SPACE=(CYL,(1,1)),DISP=(NEW,PASS,DELETE),UNIT=2314,
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//LKED     EXEC  PGM=&LINK,PARM='LIST,MAP,&LKED',REGION=&LSIZE,
//         COND=(&EL,LT,FORT)
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  SPACE=(CYL,(1,1)),UNIT=2314
//SYSLIB   DD  DDNAME=USERLIB
//         DD  DSN=&LIB,DISP=SHR
//         DD  DSN=FORTLIB,DISP=SHR
//         DD  DSN=SYS1.R19.LINKLIB,DISP=SHR
//         DD  DSNAME=SYS2.LOADLIB,DISP=SHR
//USERLIB  DD  DSN=SYS1.USERLMOD,DISP=SHR
//SYSLMOD  DD  SPACE=(CYL,(1,1,1)),DISP=(NEW,PASS,DELETE),UNIT=2314,
//         DSN=&&GOSET(&PROGRAM)
//SYSLIN   DD  DSN=*.FORT.SYSLIN,DISP=(OLD,DELETE),UNIT=2314
//         DD  DDNAME=SYSIN
//GO       EXEC  PGM=&LMOD,PARM='&GO',REGION=&CORE,
//         COND=((&EL,LT,FORT),(&LINKERR,LT,LKED))
//STEPLIB  DD  DSN=SYS1.R19.LINKLIB,DISP=SHR
//LIBUSAGE DD  DSN=SYS2.FORTUSE,DISP=SHR
//DELETE   DD  DSN=&&GOSET,UNIT=2314,DISP=(MOD,DELETE),SPACE=(80,1)
//FT01F001 DD  SPACE=(CYL,(1,2)),UNIT=2314
//FT02F001 DD  SPACE=(CYL,(1,2)),UNIT=2314
//FT03F001 DD  SPACE=(CYL,(1,2)),UNIT=2314
//FT04F001 DD  SPACE=(CYL,(1,2)),UNIT=2314
//FT05F001 DD  DDNAME=SYSIN
//FT06F001 DD  SYSOUT=A
//FT07F001 DD  SYSOUT=B
//*
//*
//*
//*          F O R T G C L G
//*
//*
//*
//*
//*
//*
```