Title
PRELIMINARY REPORT ON GRAUMP?A SYMBOLIC FORTRAN DUMP

Author
Fourt, Ed.

Publication Date
1977-03-01

PRELIMINARY REPORT ON GRUMP-- A SYMBOLIC FORTRAN DUMP

Ed Fourt

March 1977

Prepared for the U. S. Department of Energy
under Contract W-7405-ENG-48

DISCLAIMER

LBL-6762

Preliminary Report on GRUMP--
A Symbolic FORTRAN Dump*

Ed Fourt

Lawrence Berkeley Laboratory
University of California
Berkeley, California

March 1977

1.1 INTRODUCTION

This is a first report on the program GRUMP, which provides symbolic dumps of FORTRAN programs--specifically, of programs compiled by the Control Data FTN compiler--and which is now in operation at the Lawrence Berkeley Laboratory. By a "symbolic dump" we mean a printout which is generated and used like a standard octal or hexidecimal dump, to inform the user of the state of his program at some specific moment, generally the moment at which execution aborted; but which in contrast to the octal dump, refers to all variables by their FORTRAN names, and gives their values in terms of FORTRAN-style encodings, rather than simply as unlabled machine word images. A line of an octal dump might go

```
001234    17204 00000 00000 00000    00000 00000 00000 00006
```

The corresponding line of a symbolic dump might go

```
F = 1.        J = 6.
```

The usefulness of the latter form should be obvious.

This report is in no way a guide to the use of GRUMP. For that, an LBL computer user should print out a copy of the WRITEUPS subset AIDS; the

---

* This work was done with support from the U.S. Energy Research and Development Administration.

Computing Consultants can tell you how. A somewhat revised version of that user's writeup is being made available to the world at large as LBL report LBL-6764, entitled User's Guide to GRUMP, A Symbolic FORTRAN Dump. The document here is rather a report on what has been accomplished and a tentative self-evaluation of the accomplishment, together with many items of interest to the systems programmer who may be interested in putting up GRUMP at his own installation.

1.2 A SHORT HISTORY

The Lawrence Berkeley Laboratory has a history of trying to make the life of the non-specialist computer user a little easier than it is at many large scientific computing centers. An octal dumping program named DMPS which read Load Maps and was thus able to provide the user with addresses relative to the beginning of each block of a relocatable load, was made available some 10 years ago, not long after the delivery of LBL's first CDC 6600. This program, written by Bill Gage, is still more useful for our purposes than anything provided by the manufacturer today; at LBL it has been further developed, chiefly through the efforts of Roy Carlson and Tom Strong, into today's program DUMP, a very powerful octal dump indeed.

The desirability of a truly symbolic FORTRAN-oriented dump has always been obvious, however; and work was begun on one by the author, under the auspices of the LBL Systems Group, in late summer of 1975. This report is being written some 18 months later, in March 1977, when with version 4.0 GRUMP has reached what we contemplate will be a fairly stable form. It was never worked on more than half time, so the current version embodies some 9 man-months of effort.

The author spent many years as a chief programmer for various LBL physics groups; which is to say, as perhaps the only computer expert among many heavy computer users. During that time he devoted much thought to ways of making his own life easier, to mechanisms which would enable him to uncover his own and his colleagues' bugs faster, or, even better, would enable his non-specialist colleagues to uncover their own bugs themselves. GRUMP is the result; by the time work actually began on it, the situation had resolved itself into high definition in the author's mind, and work could thus proceed straightforwardly.

1.3 PROGRAM INPUTS

Three inputs are necessary for a symbolic dump--a core image giving the bit-values of all relevant computer words, a load map giving the absolute address of the first word of each relocatable block as it has been linked into the core image, and symbol tables giving the name and relative address and some of the attributes of each variable in each relocatable block. We here describe the forms in which GRUMP currently expects each of these inputs.

The core-image record read by GRUMP at LBL is a single file containing two logical records. The first is the "Exchange Package", which contains the values of all program registers and is 16 words long. The second is the core image itself. Some coding would be required to generate such a file on a standard CDC system. At LBL the control-card cracker has been modified to create such a record when certain magic control-card verbs are encountered, a small and effective ugly patch.

The load map is read in the form of a unique machine-oriented record

called a "Dbgmap", which was first designed as input to LBL's (octal) DUMP program. It seems certain that no other system generates such an object. The information contained on this record and used by GRUMP is merely this--the name and absolute starting address of every load block and whether it is a Common block or a routine block; and the name and absolute address of every entry point. It should be trivial to write a routine to get this information from the printout of any loader, e.g. from the CYBER Loader Map.

The names, locations, and attributes of all variables--i.e., the symbol table information--is currently read off the FTN compiler's output listing itself. GRUMP rewinds whatever files it is told have FORTRAN listings on them, and goes thru the files looking for an FTN "Symbolic Reference Map". When it finds one, it scoops up the information it needs. This method led to a very robust and easily-debugged piece of code; for in the developing and debugging stage, whenever a question as to GRUMP's actions arose, it was easy to track down the problem and fix it, because the file being read was not only human-readable as well as machine-readable, but also well-known to be trustworthy.

The routines that read the FTN listings have been carefully designed for ease of maximal optimization for speed--unlike the remainder of the program, which was basically designed for ruggedness and maintainability--and in fact GRUMP does run relatively fast. A full GRUMP dump often costs the user less to generate than does a full octal dump--because the GRUMP dump is much more compact and entails the formatting and printing of a much smaller bulk of information. A FORTRAN listing is a very bulky object, however, and the best way to speed up GRUMP's running time would be to enable

it to directly read a symbol table formatted by a compiler for optimal machine-readability.

Now of course no Control Data compilers turn out such a symbol table in their manufacturer-supplied versions. Furthermore, it is most desirable to avoid the problems connected with maintaining special mods to the FTN compiler. But the University of Manchester will supply one with mods to FTN that do indeed spill the symbol table to disk; these mods were written as part of their MANTRAP package, which is discussed below. Furthermore, CDC has made an announcement that they are going to develop a good Interactive FORTRAN Debugger; and the specifications for this product would seem to entail modifications to FTN such that it will indeed write out a machine-readable symbol table.

So it may be that there will soon be a standard or quasi-standard FTN spilled-symbol-table format, one that will be maintained or at least not smashed by Control Data. Should this happen, GRUMP will be modified to read same.

GRUMP also has the ability to spill the internal symbol table form it constructs, and then read it back in during a later run. This enables one to get symbolic dumps of routines which are not compiled in the same run as that in which the dump is generated. The reading of these forms is of course extremely rapid. These symbol tables are updateable; that is, a few routines may be recompiled and a large symbol table revised and re-emitted by GRUMP itself to reflect the changes.

1.4 FLOW OF PROCESSING

A GRUMP run divides itself naturally into two phases--first building up an in-core symbol table, and then generating the listing. Before describing

the two phases in turn, we note that the FORTRAN listing of the program is copiously annotated and the program itself purports to be well-constructed, and thus it itself is its own basic reference. The BLOCK DATA subprogram attempts a glossary on most variables. 99 percent of the 9200 lines of code are FORTRAN (FTN4 dialect), with the balance COMPASS. The source code contains numerous format-free coded I/O statements, which are used for debugging GRUMP itself.

The symbol table is built up as a single vector stretching back from the front of Blank Common. Notes in the BLOCK DATA subprogram contain a detailed map of its contents and methods of accessing them. First goes the information about Load Blocks and Entry Points passed on from the loader. This information is used not only for itself, but also to index the rest of the symbol table. Since we pass over the core image sequentially, the list of load blocks arranged in load order is handy for indexing the information from the compiler, which gets pasted on the end of the vector in the order in which it is presented to GRUMP. This order is not necessarily the order in which the routines were loaded.

Next, information about the individual FORTRAN blocks is read and stored. This information may be in the form of FTN output listings, where it is taken from the cross-reference map that follows each routine. It may also be taken from old GRUMP symbol tables that have been saved from a previous run. The output listings from all versions of FTN that have been put up at LBL may be read; that is, currently, FTN versions 3 thru 4.6. The program recognizes listings primarily from their page-header lines, which may vary somewhat from installation to installation. Anything on a print file that is not an FTN listing will simply be skipped over.

The map of variables in a Common Block is taken from the first instance of the block that GRUMP sees. This is not in accordance with the strict rules of FORTRAN, which allow every routine to segment the common vector as it wishes, but it is the only practical method for GRUMP to adopt. Mapping Common Blocks differently in different routines is a terrible programming practice, although a common one on primitive and unfriendly FORTRAN systems. The *COMDECK facility of CDC's UPDATE program provides a very natural and elegant way of presenting identical Common-Block definitions to all routines, however, and GRUMP both encourages and assumes its use. No user has in fact ever inquired about the method used for deciding what to call variables in Common Blocks, let alone complained about it; CDC users seem to assume that we would assume that they would be using the *COMDECK facility.

After reading FORTRAN information, and writing out the FORTRAN portion of the symbol table if such is requested, GRUMP processes any directives the user gives it. These directives may appear on the GRUMP control card itself and also on a little file of card images. Their principal use is in suppressing or shortening the listings of various load blocks; for instance, one can tell GRUMP that no more than 5 values should be printed for any array in a certain Common Block. A directive can also tell GRUMP that certain arrays contain funny-formatted data that should be dumped in octal if no normal FORTRAN interpretation of it can be made; ordinarily GRUMP would cut the printing of such an array short, figuring that it had been over-written with garbage or had never been initialized.

At this point GRUMP is ready to make its first pass over the core image. The symbol table is as complete as it ever will be, and the rest of Blank Common may now be used as a buffer for reading in large chunks of core image. This first pass is used to dig out:

- The word contained at each Entry Point (which points back to the instruction that last called the Entry Point),
- The word containing the RJ instruction that last transferred control to each Entry Point (which contains the FORTRAN line number of the CALL statement or FUNCTION reference, if the last invocation was from a FORTRAN routine),
- The contents of the word at each address that is pointed to by one of the 24 processing registers,
- Parts of the F.E.T. and F.I.T. (control tables) for each file for which there is a pointer in low core.

This may in fact entail two passes over the core image, since an Entry Point in one buffer-load may have been last called by an instruction in a previous buffer-load.

After this pass or pair of passes, the first portion of the listing is written. This consists of:

- A traceback from the routine last in execution back up the path of calls to the main program,
- A symbolic analysis of the data in the registers and the names and values of variables whose addresses are contained in any register,
- A full traceback of the latest calls to all Entry Points,
- An ultra-minimal file status report.

The symbolic register analysis is probably the most powerful item in the whole dump. The FTN compiler is obstinately conscientious about optimizing code for speed of execution, and there is no practical way to make it compile code on a line-by-line basis and reveal where the compiled code for

each line begins. Thus there is no way of automatically telling someone what line was being executed at the time the program aborted, because code for all lines between two transfer points is jumbled together in one fast-running snarl.

Thus it is often quite difficult to figure out exactly what an FTN program was trying to do when it expired. The symbolic register analysis is a great help at this point, for the addresses of sinister variables are usually sitting in A-registers and GRUMP can immediately tell you their names and values. This is almost as useful as the information one gets from the old RUN or new MNF FORTRAN compilers, which can tell you which line of code it was that blew you up.

The file report currently tells one very little--the external and internal names of the file, whether or not it had ever been used, and the octal value in the F.E.T.'s request/response field if it in fact had been used. This is sufficient to its purpose, however, which is to uncover a very puzzling class of errors--namely, the errors caused by the connecting of an internal logical file to the wrong actual external file. CDC has a convention for changing actual-file names at execution time by positional arguments on the control card which is both powerful and dangerous, and since the first I/O reference to a previously undefined actual-file causes that file to be created on disk, it is fairly easy to make a mishmash of your I/O and not be able to figure out what went wrong. But the simple printing of logical and actual names for all files makes this error easy to uncover.

We then make a second pass over the core image in order to print out the values of all variables. It would in fact be possible to get by with a

single core-image pass, but the current way of doing things will greatly reduce the core that GRUMP requires when it is overlaid or segmented. It would not be a particularly difficult chore to change over to a one-pass operation, but this will probably not be done, because it is felt that the current most-needed improvement is reduced core requirements, and not reduced I/O.

If a block is short enough to fit entirely within GRUMP's core-image buffer, then the variables in it will be printed in alphabetical order; otherwise, they must be printed in load order. The variable listing is written as formatted print-lines to a temporary file; and when this is finished, an attempt is made to read the entire temporary file back into Blank Common at once. If this is successful, then the blocks are copied out of Blank Common to the ultimate print file in alphabetical order; if Blank Common is not of sufficient size to hold the entire variable listing, then this listing is merely copied to the print file in the order it was created; i.e., in load order. The net result of all this alphabetization is that it is extremely easy to find your way around a GRUMP variables listing.

GRUMP will in general print out the values of all elements of an array, treating the array as a one-dimensional vector because in fact no dimensionality information is obtainable from the FTN listing beyond the bald fact that the variable is not scalar.

Two conventions are used in an attempt to keep the bulk of the printout within bounds. One is the DATA-statement convention for indicating consecutive identical values in a vector, that is the convention of saying '<count>* <value>'. Thus if the 20 elements of array AR are all identically zero, GRUMP will report

```
AR (20)  =  20* 0.
```

The other is to stop printing values when five consecutive elements are found containing uninterpretable values; the assumption is that the array is in fact uninitialized, and filled with leftover executable code, or else has been overwritten with garbage. At LBL the former is particularly frequent with large arrays that sit in Blank Common, for Blank Common is generally not initialized and when execution begins it contains the executable code of the loader, which is of no interest to the user. In these cases a message saying "block dissolves into weirdness, no more interpretations" appears, along with an appropriate diagnostic, and printing of values halts.

It turns out to be quite easy to tell from the form of a word of core whether it represents a floating-point number, an integer, a Hollerith (character) value, or garbage. There are few ambiguous cases, given the use of some obvious heuristics--all floating-point must be normalized and of a 'reasonable' size, no integer may have absolute value greater than 2**48-1, Hollerith must contain no embedded zero-bytes or characters with octal value greater than 60_8. One of the few exceptions of note is the Hollerith value 6LOUTPUT, , which is also a perfect 20.255... In the few cases where GRUMP is able to make more than one good interpretation of a value, all interpretations are printed, enclosed in brackets.

Control Data computers are able to generate what are called Indefinite and Infinite forms. Their essential purpose is to blow up a program as soon as it attempts to use or succeeds in calculating (depending on the machine) one of them, in a generally successful attempt to make a program abort as soon as a problem becomes apparent. GRUMP takes special care with such forms, since they are usually entwined with program problems.

Many CDC systems, including LBL's, automatically preset all otherwise

uninitialized variables to a special negative-indefinite-plus-address form at load time; this type of value will cause an immediate abort when used as a floating-point operand or as a subscript, or when by horrible mischange it is executed as an instruction, and since it contains its own address each value is unique to its variable element. GRUMP will print such values sitting at the proper address as simply ** or *** ; if the value should contain the wrong address, a not-particularly-helpful note is printed to that effect.

This values-of-variables information is not as helpful in tracking down the immediate cause of a program abort as is the printout from the first pass. It has the great advantage, however, of being extremely readable and compact, and of often pointing out to the auditor a half-dozen further bugs, many of which would have caused incorrect answers to be calculated without causing any machine interrupts at all. Experience has shown this information to be invaluable; even more than the reduced time it takes to find the immediate cause of a program abort, it is this ability to show the programmer the conditions that will lead to future aborts and the even more abominable future incorrect results, that can so greatly reduce program development time when GRUMP is used.

1.5 SOME NOTES ON OUTPUT

Selected pages from some sample GRUMP printout are included in an appendix. We point out a few of its features.

We may note that the immediate problem is instantly obvious--the user has forgotten to initialize the variable XTARG in Common /EXTRA/, and an attempt to use it in subroutine POINTS has blown him up. In casting our eyes further on down the listing, however, we immediately encounter another

suspicious circumstance--most of the variables in commons /ADJS/, /CHI/, and /CR/ are also uninitialized. If the auditor is fairly knowledgeable about the program he is using, which is usually the case, these facts may tell him about more bugs that will inevitably bite him in the future if he doesn't fix them now. In this case, in fact, the uninitialized state of /ADJS/ is indicative of programmer error; and another bug is smashed before it can really be born.

We note that the traceback is segregated into two columns, the first for calls for which the FORTRAN line number is known and the second for those for which it isn't. Since in fact the second column contains almost exclusively calls of one system routine upon another, which are of almost no interest, this nicely reduces the bulk of information that the user has to plow through. We also note the tiny status-of-files report, here simply conveying to us the information that all is well in this department.

The listing was in fact some nine pages long, and we also reproduce the next-to-last page here. By examining the variable FEEØ in common /TS/, we can get a good idea of how GRUMP handles array printout in a normal case. We note that the array is 48 words long. The first 4 elements are uninitialized; the values of elements 5 thru 9 also print out on the first line. The (10) preceeding the second line of values indicates that the first value printed on the line is for element 10; i.e., for FEEØ(10) if we consider it a vector; and similarly, the first value in the third line is for FEEØ(16). Since the last 28 elements are all uninitialized, the printout only occupies 3 print lines.

Further down we note that the variables X and Y in routine XATAN have no values printed out for them, but are simply labeled as parameters, which

in fact they are. GRUMP currently makes no attempt to print out the values of arguments to formal parameters, even in the case of routines which stand in the direct line of calls to the routine in which execution halted, which are the only FTN routines for which such values are retrievable.

This is a flaw. It would be nice to print out argument values in the cases for which they are retrievable, and someday it may well seem worthwhile to put forth the effort necessary to enable this. It will certainly not happen as long as GRUMP is reading FTN Symbolic Reference Maps, however, for they don't supply enough information--they don't tell you what the ordinal of a parameter is; i.e., whether it's the first, or second, or third parameter of a routine that requires three arguments. The felt need for this improvement is not huge but far from infinitesimal.

Looking on down at the printout for Blank Common, we find GRUMP dealing with a more recalcitrant set of arrays. Their images start off with nice values, but swiftly degenerate into what is in fact left-over Loader code, which is sitting in the unused portions of Blank Common. GRUMP is able to realize that it in fact is dealing with garbage and cuts itself off, printing a message which in fact correctly analyzes the situation.

It is probable that after the appearance of a half-dozen or so ugly Blank Common dumps like the present one, the user would be motivated to do a little digging in the documentation and discover the parameters necessary to get Blank Common preset to the same innocuous negative indefinites as the rest of core is. We may note that in the meantime GRUMP's output here, while hideous, is at least accurate.

The arrays AAA and AX are EQUIVALENCEd. That is why the two names appear paired on the first line printed for Blank Common and why the line

AX SEE AAA

appears below.

This ten-page output provided a complete symbolic image of the state of all variables at a moment in the execution of a physics code that occupies some 70K (octal) words of core. This is about an average length for a full GRUMP dump, which for all but tiny beginner programs usually occupy 5 to 20 pages. It is extremely unusual for a GRUMP dump to require more than 25 pages; i.e., more than 1500 lines--at least at LBL, where no program longer than 170K (octal) words can fit into the small core memory of the main 7600 computer.

1.6 EVALUATION AND COMPARISON

We may now articulate and evaluate some basic features of GRUMP's design.

1. Printout is compact but complete. This was thought to be a most important principal when work began, and is still seen as such. There is nothing more frustrating than having an "intelligent" dumping program decide that you don't need a particular piece of information when in fact you do. GRUMP attempts to print out every single item of data in the program unless it seems almost certain to be misplaced executable code, and it will even print in octal stuff that looks like garbage if it is requested to. Also, in the default case, all elements of all arrays are printed, although it is easy for the user to reduce or suppress the printing of large useless blocks.

But in order to be so complete, GRUMP has to compact its printout as much as possible. Thus, for instance, it does not explicitly tell you that a value is for a variable typed Real or Integer or Logical or whatever, but simply prints out floating-point values with a decimal point and fixed-point

ones without, shows Booleans as .TRUE. or .FALSE. , etc. In the case where an element whose only type is Real contains a non-zero value with zero exponent, GRUMP will print both the floating point 0. and the non-zero integer, bracketing the pair; and similarly for other values corresponding to no extant declaration of their containing variable. The futile printing of strange values as gargantuan integers, or of good floating point as bad Hollerith, is avoided. Also, variables local to routines that were never called are not printed out at all. A barrage of trivial information about declarations is liable merely to discourage the user; GRUMP shoots for elegance rather than overload.

2. It is economical. At LBL's current computer recharge rate a GRUMP dump costs between 25 cents and $1.50 to produce, which is within reason although not infinitesimal. At 1-1/2 cents per page, the printing shouldn't add more than another 25 cents; a full octal dump of 100 pages, in contrast, costs $1.50 for paper alone, plus processing charges which may well be higher than GRUMP's. The shortness of the listing cuts turnaround time way down in shops where short jobs print first; and the completeness of the listing maximizes the number of bugs squashed per run, thus cutting down program development time and number of debug runs.

3. It attempts to unveil the state of a program without attempting to diagnose that state. It is thought that a ten-page "how to debug" writeup (currently unwritten) is necessary and should be sufficient to enable anyone to diagnose most of his own problems. It is also thought that a concentration on the immediate problem is less profitable than a consideration of the program as a whole, as an object riddled with problems, many of which will neither cause an abort nor be diagnosable. The author's experience with

diagnostic dumps has not been particularly pleasant; for all but the simplest problems, i.e. uninitialized variables and division by zero, they often mislead. Also they often over-diagnose; i.e., inform the user over and over again of things which are in fact not errors. GRUMP avoids these problems currently via the simple expedient of printing no diagnostics whatsoever, which is probably overdoing things; but hopefully GRUMP will always err a little on the side of terseness.

4. The output is not ugly. The hardest part of the building of GRUMP was preventing its printout from taking on a hideous, shrapnel appearance. An ugly dump will be read as little as possible; GRUMP attempts to invite the user in for a further look around, to make it pleasant for him to amble through the state of his program and find out what else beyond the immediate problem is wrong with it. Non-ugliness ranks with compactness as one of the overriding fetishes that controlled the design and development of GRUMP.

5. It is easy to begin using and easy to tune with experience. The first-time user simply has to insert a control card saying

```
GRUMP.
```

in his deck, and something pretty close to what he really wants will appear automatically. There are many execution-time arguments which may be used to customize one's dump, however; they are enumerated in the User's Manual. The philosophy has been that any feature requested will be inserted, but as a parameter with a sensible default.

The author knows of two other somewhat similar symbolic dumping programs, the University of Manchester's MANTRAP package, and the University

of Michigan's DUMP. We will draw our comparisons only with MANTRAP, since it also runs on Control Data hardware. (Michigan's program runs on IBM-compatible machines.)

MANTRAP is a very nicely designed package indeed, and this author is happy to report that he feels that it serves a completely different programming environment than does GRUMP. MANTRAP is highly diagnostic-oriented and extremely verbose; it is oriented towards small student programs that embody gross misconceptions about how it is that FORTRAN actually works. It seeks to explicitly teach computing science; it tries to impress upon the student the view of a program as tree-structured and consisting of a heirarchy of modules that talk back and forth via parameters passed in calls upon routines.

This view of computing is presumably a fine one to impress upon the young, but in fact has little to do with the structure of most big old FORTRAN programs. They are in fact, and presumably unfortunately, a blurry web of references to global variables residing in a mishmash of Common Blocks; and the best and only way to clarify their structure is to <u>alphabetize</u> everything possible. This is not because FORTRAN users are dummies or because scientists and engineers haven't studied enough academic computing, but because large scientific codes live long, long lives and get modified by many, many people. GRUMP's basic use is in modifying big old codes, for that in fact seems to be the main business done at the Lawrence Berkeley Laboratory Computer Center. For an environment of beginning students writing beginner programs, MANTRAP is undoubtedly a better product.

1.7 PROSPECTS

GRUMP should be in a fairly stable state now; it is thought to embody most necessary features. It is hoped that the following can be added when time allows:

- Retrieval and printing of argument values where this is possible
- Analysis of the contents of RA+1 (the area for communication with the monitor) when they are non-zero
- A line of diagnostics dependent upon the immediate cause of abort--mode error, time limit, user abort, or whatever
- Reducing core requirements. The current unsegmented form contains some 34K (octal) words of instructions and Labeled Common. By getting rid of FORTRAN library references and using the segmentation loader, at least 12K of this should be recoverable. At least 20K, and preferably more than that, are additionally necessary for Blank Common.
- Perhaps the ability to dump 7600 LCM
- Reading of machine-oriented FTN symbol tables when a standard form for same seems to exist
- Reading of machine-readable symbol tables put out by the MNF FORTRAN compiler. This is actually being installed at the time this report is being written.
- Elimination of FORTRAN Formatted I/O in favor of more efficient and smaller subroutines.

GRUMP is ready to begin running on systems other than LBL's. This system is locally written and unique, but in fact it quite closely resembles the standard 6000-series KRONOS and NOS systems, and transfer to those should

be a mere chore. Transfer to standard 7600 SCOPE 2. will be somewhat more difficult, though hardly a challenge, because some I/O that explicitly calls CIO will have to be rewritten for the Record Mangler. The author would be interested in hearing from users at other sites that would be interested in adapting GRUMP to their local system and making the necessary mods available to the world at large.

GRUMP has only recently been put up on LBL's 6600 system. Therefore, no detailed timing trials have yet been made; only the times required for entire runs are currently known. This is because 7600 timing histograms are tedious and expensive to gather, because a software simulator must be used; but 6600 timing trials are trivial and cheap, because of the system architecture.

Thus there are undoubtedly some worthless loops lingering undiscovered within the bowels of GRUMP, areas of code that sop up time unnecessarily, and one of our next projects is to uncover them. We hypothesize that the biggest time-burners are the listing-reading routines, which are designed to permit easy optimization but have not in fact been optimized, and the FORTRAN ENCODE statement, which is used to translate most variables into printable format. But no doubt big blunders are lurking there too....

Special thanks to Jerry Knight, for insisting that the output be pretty, and to Bill Johnston, for insisting that it be correct.

```
         ***   G R U M P   D U M P   ***             LPL VRS  5.3                    20 AUG 77 12.52.18  LBL/BKY


PROGRAM HALTED AT ADDR 000133 IN ROUTINE  POINTS        (ABSOLUTE ADDR 000653)
        CALLED FROM  TRACKS     AT LINE  101      (RELATIVE ADDR 000030)
        CALLED FROM  HOWL       AT LINE  113      (RELATIVE ADDR 005340)
                 ***   MAIN PROGRAM   ***


                        CONTENTS OF REGISTERS

                   A REGISTERS (CONTAIN ADDRESSES)
          ADDRESS    . . . . . . . . . . . . . . . . . . . . . .   CONTENTS
   A0   053005  ->  000052 IN -TRACKS-          --    00000000000000053013   = 22027
   A1   001432  ->  000712 IN -POINTS-          --    17166100313002702624   = .38286  OR  'ONI YXB+VT'
   A2   065647  ->  000015 IN SINCOS=           --    04000006500000000000   = 'D  F/      '
   A3   065721  ->  000067 IN SINCOS=           --    17147235645603410600   = .11419  OR  'OL<2*,C6F '
   A4   000126  ->  XTARG   IN /EXTRA/          --    60000000000000400126   = ***   UNINITIALIZED   ***
   A5   000311  ->  RAD     IN /MAKE/           --    17227105562441453542   = 7.1362  OR  'OR'E,T6+27'
   A6   000267  ->  XOC     IN /MAKE/           --    17770000000000000000   = ***   CALCULATED POSITIVE INDEFINITE   ***
   A7   000323  ->  PBETA   IN /MAKE/           --    17327614533513615236   = 1990.3  OR  'OZ¬L$2K[)3'

                   X REGISTERS (USUALLY CONTAIN DATA)
   X0   17226475142626040354  = 6.6195  OR  'OR≠≥LVVDC='
   X1   17166100313002702624  = .38286  OR  'ONI YXB+VT'
   X2   00000000000400000650  = 67109288
   X3   17203553562600665621  = 'OP2$,V v,Q'
   X4   60000000000000400126  = ***   UNINITIALIZED   ***
   X5   17227105562441453542  = 7.1362  OR  'OR'E,T6+27'
   X6   17770000000000000000  = ***   CALCULATED POSITIVE INDEFINITE   ***
   X7   17770000000000000000  = ***   CALCULATED POSITIVE INDEFINITE   ***

                   B REGISTERS (MISCELLANEOUS)
   B0   000000  = 0
   B1   000650  = 424,    ->  000130 IN -POINTS-           --   51500003115140000126   = -9.29638+136  OR  '(/ CI(5 AV'
   B2   000001  = 1
   B3   777767  = -8
   B4   777745  = -26
   B5   074017  = 30735,  ->  PCMEAS(13) IN //             --   00000000000000000000   = 0
   B6   000004  = 4
   B7   000005  = 5


                        TRACEBACK OF LATEST CALLS TO ROUTINES

        BLOCK.....ENTRY.....FROM......LINE......REL ADDR                BLOCK.....ENTRY.....FROM......REL ADDR
        DECODE=   DECODI.   REED       240      000107                  ALOG      ALOG.     SAGE      000105
        DECODE=   DECODR.   REED       240      000107                  ATAN      ATAN.     XATAN     000026
                  EOF       REED       239      000105                            CIO=      ERM$$     000575
                  GO        SAGE       300      000333                  SINCOS=   COS.      POINTS    000127
                  HOWL      ***   MAIN PROGRAM   ***                    FMTAP=    FECAP.    OUTC=     000130
                  INITIZE   SAGE       180      000134                  OUTCOM=   FECCHR.   KODER=    000056
        INPC=     INPCI.    REED       236      000103                  COMIO=    FECC63.   OUTC=     000165
        INPC=     INPCR.    REED       236      000103                  FORSYS=   FECOPE.   OUTC=     000105
        OUTC=     OUTCI.    PRAM        41      000033                  INCOM=    FEIFST.   KRAKER=   000357
        OUTC=     OUTCR.    PRAM        41      000033                  FLTOUT=   FEOFAL.   KODER=    000307
                  POINTS    TRACKS     101      000030                  FLTOUT=   FEOSCA.   KODER=    000300
                  PRAM      HOWL       109      005334                  Q8.IO.    FTNRPV.   HOWL      005320
                  RAND      GO         231      000463                  GETFIT=   GETFIT.   OUTC=     000023
                  REED      HOWL        99      005325                  KRAKER=   KRINIT.   INPC=     000132
```

00104807662

RX	SAGE	[illegible]	[illegible]
SAGE	HOWL	111	005336
SAME	REED	218	000067
SLAP	REED	351	000256
TRACKS	HOWL	113	005340
WT	SAGE	309	000342
XATAN	POINTS	171	000105

	[illegible]	[illegible]	[illegible]
SINCOS=	SIN.	POINTS	000107
SQRT	SQRT.	POINTS	000115
ERM$$	STATS.	ERM$$	000562
	SYS=	CIO=	000005
UZERO..	UZEROOO	Q8.IO.	000074
	WTH=	ERM$$	000406
XTOI=	XTOI.	GO	000057
ERM$$	ZZ.GET	INPC=	000161
ERM$$	ZZ.OPE	FORSYS=	000705
ERM$$	ZZ.PUT	OUTC=	000176

LIST OF ENTRY POINTS THAT WERE NEVER CALLED

ABNORM.	ATAN2	DCB=	FECEE.	FEIFSC.	FEOR1F.	INPBR.	NAME.	RDW=	STRAY	SYS2=	ZIPSE
ABORT	ATAN2.	DEDX	FECFMT.	FEIFSG.	FEOR10.	IOERR.	NESTLE	RDX=	STREAM	SYS=6	ZZ.CHK
ABORTC	BFN.	DET.	FECFMU.	FEIGNC.	FEOXFL.	ITOJ.	NGONG	REWIND.	SYP=1	TAN.	ZZ.CLO
AB1.	BLANK	DRIFT	FECJP=	FEINUM.	FEOZRO.	KODWRT=	ORDR	RINGER	SYP=3	TORTURE	ZZ.EOF
ACOS	CBD.	ENCODI.	FECLP.	FEISBL.	FINALE	KOJPT.	OUTBI.	ROTN	SYP=4	TRIGGER	ZZ.EOR
ACOS.	COKPCT	ENCODR.	FECMSK.	FEOAFM.	FLLCM.	KOOL	OUTBR.	ROTTEN	SYP=5	TRIGSAV	ZZ.REW
ADD	CHISQ	END.	FECNAP.	FEOBLS.	FLOW	KOREP.	PLOP	RUNGE	SYSAID=	WHERE	ZZ.SKP
ALOG	CIRC	ERRSET	FECPRT.	FEOCNV.	FLSCM.	LCB=	PRETSUM	SCAT	SYSAID.	WHIRL	
ALOG10	CLSLNK.	EXIT	FECRP.	FEOEOV.	FUNWT	LDRUSX=	Q8.IO.	SHUFFLE	SYSARG=	WNB=	
ALOG10.	CLUTCH	EXP	FECV.	FEOEXP.	GAUSS	LEONARD	RANDOM.	SIN	SYSEND.	WOBBLE	
ANORM	CMF=	EXP.	FEIBLK=	FEOI.	GOTOER.	LINLIM.	RATE	SLACOUT	SYSERR.	WTL=	
ASIN	COD.	FAN	FEIBLK.	FEOL.	GO	LODGEN	RAZ	SQRT	SYSE3.	WTW=	
ASIN.	COS	FAR	FEIERR.	FEONTL.	HFIELD	MAXWT	RCL=	STOP.	SYST1A.	WTX=	
ATAN	COSMO	FECBUG.	FEIEXP.	FEORND.	INPBI.	MORGUE.	RDL=	STRAIT	SYS1ST.	XTOY.	

STATUS OF FILES

EXT NAME....	INT NAME....	STATUS	EXT NAME....	INT NAME....	STATUS
INPUT	INPUT	000013	OUTPUT	OUTPUT	000027
RALPH	RALPH	UNUSED	CYRIL	CYRIL	UNUSED
FOURVS	FOURVS	UNUSED	DHOWL	DHOWL	UNUSED
OUTPUT	DEBUG	000027			

VALUES OF VARIABLES WHEN PROGRAM HALTED

```
/ADJS/          VARIABLES IN COMMON /ADJS/
    RFRC      = **                WANDZO (16) = 16* ***        XRT (8)   = 8* ***          XTR (8)   = 8* ***
    YRT (8)   = 8* ***            YTR (8)   = 8* ***

/CHI/           VARIABLES IN COMMON /CHI/
    CHIXY (50) = 50* ***          CHIZ (50) = 50* ***

/CR/            VARIABLES IN COMMON /CR/
    CHI       = **                ERROR     = 1.00000E-12      M         = 0                 NCYC      = 0
    R         = **                XO        = **               YO        = **                ZO        = **
    Z1M       = **                Z2M       = **

/CYLPRM/        VARIABLES IN COMMON /CYLPRM/
    CYLDF [ CYLDF DF ] (16) = 16* ***                          CYLDR SEE CYLX2               CYLDTW [ CYLDTW DTW ] (4) = 4* ***
    CYLDO [ CYLDO DO ] (4) = 4* ***                            CYLD1 [ CYLD1 D1 ] = **
    CYLP [ CYLP CYLX1 PHIO ] (16) =                  6.26000E-02,      7.41600E-02,      .14504,           .13449,
        (5)  6.52600E-02,      8.05000E-02,      .15920,           .14472,           7.25800E-02,      8.95300E-02,
       (11)  .17552,           .16049,           .10455,           .12011,           .23934,           .26244;
    CYLR [ CYLR RNOM ] (16) =  1.3608,           1.3511,           1.3414,           1.3314,           1.1321,
```

```
CYLS [ CYLS SIZE ] (4) =      2.6750,                [illegible]                [illegible]                [illegible]:
CYLSET    = **
CYLTP [ CYLTP TP ] (4) =      -4.20000E-04,          4.00000E-05,               1.50000E-04,               4.10000E-04:
CYLTR [ CYLTR TR ] (4) =      2.89000E-03,           3.65000E-03,               -1.60000E-04,              -3.15000E-03:
CYLTY [ CYLTY TY ] (4) =      2.90000E-04,           4.70000E-04,               1.10000E-04,               6.10000E-04:
CYLX [ CYLX XO ] (4) =        5.81000E-03,           6.72000E-03,               6.53000E-03,               7.30000E-03:
CYLX1 SEE CYLP                CYLX2 [ CYLX2 CYLDR DR ] = **
CYLY [ CYLY YO ] (4) =        -1.50000E-03,          -1.18000E-03,              -9.60000E-04,              -5.50000E-04:
CYLZ [ CYLZ ZO ] (4) = 4* 2.48000E-02                                           DF SEE CYLDF               DR SEE CYLX2
DTW SEE CYLDTH                DO SEE CYLDO                                      D1 SEE CYLD1               PHIO SEE CYLP
RNOM SEE CYLR                 SIZE SEE CYLS                                     TP SEE CYLTP               TR SEE CYLTR
TY SEE CYLTY                  XO SEE CYLX                                       YO SEE CYLY                ZO SEE CYLZ

/DCHINFO/       VARIABLES IN COMMON /DCHINFO/
CHTYPE (30) = 2* 'PROP CH   ',                        8* 'SPARK CH  ',    20* ***;
CLOCK     = **                ICHOFPT (500) = 500* ***                    IDAZM (1500) = 1500* ***
IDCH (30) = 1,                2,                      3,                  4,                  5,                  6,
    (7)  7,                   8,                      9,                  10,                 20* ***:
NDAZMS    = **                NDCH      = 0
NDPTS (20) = 0,               19* ***;
NWIRES (30) = 30* ***         PHIOFO (30) = 30* ***                       SPEED     = **                      TWIST (30) = 30* ***

/ENDCAP/        VARIABLES IN COMMON /ENDCAP/
ECPHIO (4) = 4* ***           ECSIG (4) = 4* ***                          IECP      = **                      LIBEAM    = **
MAXECPS   = 4                 MXECPET   = 100                             MXECPTK   = 10                      NECPPTS   = 0
NNECPS    = 0                 RECPMAX (4) = 4* ***                        RECPMIN (4) = 4* ***                ZECP (4) = 4* ***

/EXTRA/         VARIABLES IN COMMON /EXTRA/
GKM       = .51100            GKP       = 0.                              GKPCT     = **                      IEVENT    = 1
IFOURVS   = 0                 IRUN      = 0                               IRUNCTR   = 1                       ITRYTOT   = 0
ITRY1     = 0                 ITYPE     = 0                               NEVTOT    = 5                       NFINAL    = **
NTRYS     = 1                 XIR       = **                              XTARG     = **                      YIR       = **
YTARG     = **                ZIR       = **                              ZTARG     = **

/FILES/         VARIABLES IN COMMON /FILES/
CYRIL     = 'CYRIL'           DHOWL     = 'DHOWL'                         FOURVS    = 'FOURVS'                INPUT [ INPUT NTPIN ] = 'INPUT'
NTPIN SEE INPUT               NTPOUT [ NTPOUT OUTPUT ] = 'OUTPUT'                                           OUTPUT SEE NTPOUT
RALPH     = 'RALPH'

-GO-            VARIABLES LOCAL TO ROUTINE -GO-
AA [ AA II ] = **
A1 (8)    = -320.23,          795.14,                 1801.6,             2000.0,             -320.23,            795.14,
    (7)  1801.6,              2000.0:
A2 (8)    = 320.23,           -795.14,                -1801.6,            2000.0,             320.23,             -795.14,
    (7)  -1801.6,             2000.0:
A3  (PARAMETER...)            A4  (PARAMETER...)                          BCOS      = 'BCOS     '             BDEL      = 'BDEL     '
BPOL      = 'BPOL     '       B1 [ B1 I1 ] = **                           B2 [ B2 I2 ] = **                   B3 [ B3 I3 ] = 0.
B4 [ B4 I4 ] = 0.             D3SQ      = **                              EA        = **                      FMAX      = **
FMIN      = **                FN        = .50000                          FOURPI    = 12.566                  FPISQ     = 157.91
GAGE      = **                GO2       = **                              HCOS      = 'HCOS     '             II SEE AA
I1 SEE B1                     I2 SEE B2                                   I3 SEE B3                           I4 SEE B4
J         = 5                 ONE4V  (PARAMETER...)                       PI (477) = 4* ***                   PM        = **
Q         = 1995.2            QI        = **
R (3)     = -.16050,          .39853,                 .90300:
RGE       = **                RHO       = **                              SINE      = .42964                  TCOS      = 'TCOS     '
TDEL      = 'TDEL     '       TMAX      = **                              TMIN      = **                      TPOL      = 'TPOL     '
TWOPI     = 6.2832            TWO4V  (PARAMETER...)                       TX        = **                      TY        = **
UMAX      = **                UU        = **                              U1        = 139.00                  U2        = 139.00
Z         = .50000            ZZZ       = **

/HEAD/          VARIABLES IN COMMON /HEAD/
FITB      = 1.0000            LVAC      = **                              UX        = **                      UY        = **
UZ        = **                XIN       = **                              YIN       = **                      ZIN       = **
```

```
      NP (5)    = 5* ***
      UU (495) = 495* ***

/STRAND/        VARIABLES IN COMMON /STRAND/
      IRAND     = 0

/SWITCH/        VARIABLES IN COMMON /SWITCH/
      ISW       = 0

-TRACKS-        VARIABLES LOCAL TO ROUTINE -TRACKS-
      I         = 2                 J         = 4                 K         = 5

/TRIGGER/       VARIABLES IN COMMON /TRIGGER/
      IFIRED    = **                ITRIG (48) = 48* ***          ITRIGCH   = 0                 LTRIG     = **
      MAXTRIG   = **                MINTRIG   = **                NTRIGS    = **                TOPPLE    = **
      TRIGHLN   = **

/TS/            VARIABLES IN COMMON /TS/
      CAN       = 1.0000
      FEEO (48) = 4* ***,           .26244,             .23934,             .12011,             .10455,             .16049,
         (10)  .17552,              8.95300E-02,        7.25800E-02,        .14472,             .15920,             8.05000E-02,
         (16)  6.52600E-02,         .13449,             .14504,             7.41600E-02,        6.26000E-02,        28* ***;
      HAFSIZE (24) = .25400,        .40640,             2* 1.1025,          2* 1.1000,          2* 1.2090,          2* 1.3375,
         (11)  14* ***;
      HAFWIR    = .25000            NPHIO     = 20                NSIZE     = 10
      SIZESQ (24) = .25806,         .66064,             2* 4.8620,          2* 4.8400,          2* 5.8467,          2* 7.1556,
         (11)  14* ***;
      SIZZ (24) = .50800,           .81280,             2* 2.2050,          2* 2.2000,          2* 2.4180,          2* 2.6750,
         (11)  14* ***;
      TANSKEW (4) = 4* ***

/USER/          VARIABLES IN COMMON /USER/
      NUSER     = 0                 YOUSER (56) = 56* ***

/WOBBLE/        VARIABLES IN COMMON /WOBBLE/
      BETA      = .99758            COSFL     = .42964            FLOP      = 1.1267            PMASS     = 139.00
      PXYZ      = 1995.2            PXYZSQ    = 3.98068E+06       RADSQ     = 50.925            SINFL     = .90300
      XYMOM     = **

-WT-            VARIABLES LOCAL TO ROUTINE -WT-
      A1  (PARAMETER...)

-XATAN-         VARIABLES LOCAL TO ROUTINE -XATAN-
      A180      = 3.1416            A270      = 4.7124            A360      = 6.2832            A90       = 1.5708
      DANGLE    = 1.1267            PUTZ      = 2.1018            X  (PARAMETER...)             XATAN     = 1.1267
      Y  (PARAMETER...)

/ZZZ/           VARIABLES IN COMMON /ZZZ/
      PMP       = **                RHO2      = **                SIG2VX    = 0.                ZBGN      = **
      ZRAT      = **

//              VARIABLES IN BLANK COMMON //
      AAA [ AAA AX ] (100) =        12* 0.,             (UNNORMALIZED) 1.06688+242,             (UNNORMALIZED) 5.78204-116,
         (15)  (UNNORMALIZED) 7.26082+280,             'ISIGN$',           [ 0. 262145 ],      4000000400000100000008
.....BLOCK DISSOLVES INTO WEIRDNESS, NO MORE INTERPRETATIONS.....
.....PROBABLY LEFT-OVER LOADER CODE IN BLANK COMMON.....
      AX SEE AAA
      AY (100) = 4000.0,            0.,                 3* 139.00,          (UNNORMALIZED) 1.06688+242,             (UNNORMALIZED)
         (7) 1.95240-115,           (UNNORMALIZED) 7.26082+280,           'ITOX$.',           [ 0. 262145 ],
         (11) 4000000400000100000008
.....BLOCK DISSOLVES INTO WEIRDNESS, NO MORE INTERPRETATIONS.....
.....PROBABLY LEFT-OVER LOADER CODE IN BLANK COMMON.....
      AZ (100) = (UNNORMALIZED) 7.26082+280,            'KODER',            [ 0. 262145 ],      4000000400000100000008,
         (5)  'H',                  (UNNORMALIZED) 5.50919E-79
.....BLOCK DISSOLVES INTO WEIRDNESS, NO MORE INTERPRETATIONS.....
```

TECHNICAL INFORMATION DEPARTMENT
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720