E

N71-36389

Preceding page blank

NASTRAN INSTALLATION: IMPLEMENTATION STEPS AND POSSIBLE PROBLEMS ENCOUNTERED

By Howard E. Dielmann

Computer Sciences Corporation

SUMMARY

NASTRAN, from its inception, was designed to operate on several diverse computer systems. It is currently installed and operating on the CDC 6600, the IBM 360, and the UNIVAC 1108. This paper discusses the steps found by CSC to be necessary in installing NASTRAN on a computer system and the possible obstacles that might be encountered in undertaking NASTRAN installation. Reference is made to actual problems that arose during installation on the above machines. With a knowledge of what has happened to date in setting up NASTRAN, the future user will be better able to cope with and understand the implications of installing NASTRAN on his computer.

INTRODUCTION

The NASTRAN program had as its primary directive to design and develop NASTRAN in such a manner that it could be easily implemented on several different machines. These machines were the IBM 7094 under IBSYS, the UNIVAC 1108 under EXEC II and EXEC 8, the IBM 360 under OS, and the CDC 6600 under SCOPE. The problems associated with machine dependence were given considerable attention during the initial phases of NASTRAN's development. An attempt was made to utilize a subset of FORTRAN IV that was compatible with the specified machines (reference 1), and to avoid dependencies on machine characteristics such as 36-bit words or 6-bit characters. Despite the planning devoted to avoiding dependencies, many obstacles arose whenever NASTRAN was implemented on a computer system. These obstacles generally occurred because of the enormous size of NASTRAN,[1] its extreme and varied demands on the operating system, or the lack of capabilities of the particular operating system involved.

[1] NASTRAN consists of over 700 subroutines and comprises some 145,000 cards. The compiled object code contains approximately 900,000 words, and the absolute or executable program consists of over 1,800,000 words (IBM 360 statistics).

Early in the development of NASTRAN, the hardware capabilities and to some extent, the software capabilities of the second generation IBM 7094 and UNIVAC 1108 EXEC II were taxed excessively, and NASTRAN evolved into a third generation system. The third generation systems are better equipped to meet NASTRAN's hardware requirements, but no operating system to date has been able to sufficiently satisfy all of the software requirements of NASTRAN. These third generation operating systems are not designed to give to any one application program the total resources that NASTRAN requires (core, files, etc.) without limiting the execution of NASTRAN or affecting the operation of the computer system.

The intent of this paper is to identify both the steps and the obstacles that might be encountered in installing NASTRAN. The particular problems encountered in adapting NASTRAN to other computer systems cannot, of course, be addressed by this paper, since these will vary from one computer system to another.

However, a general observation can be made that will apply to the implementation of NASTRAN regardless of the particular computer system involved. The observation is this: that a comprehensive and flexible operating system greatly aids in the installation effort. Many of the obstacles we encountered in installing NASTRAN involved services related to the operating system, such as library editing, compiling, loading, and I/O control. If the proper facilities either were not available, or were insufficient, programs had to be written to provide these services. We therefore feel that the following features of an operating system are either necessary or desirable: (1) a source and object library editing capability, (2) a standard and efficient FORTRAN IV compiler, (3) a sophisticated loader capable of handling large overlays, (4) an extensive control language that allows multiple file definitions, dump specifications, etc., and (5) macro-level I/O calls.

This paper is presented in the general order of the steps employed in installing NASTRAN:

1. Hardware evaluation

2. Software implementation

    a. Source language creation

    b. FORTRAN compilation

    c. Source and object library updating

    d. Machine-dependent deck conversion

    e. Absolute program creation

f. NASTRAN execution

g. Tailoring of NASTRAN to the computer system

HARDWARE EVALUATION

The initial question to be resolved when considering the installation of NASTRAN on a computer system is whether the hardware configuration will support NASTRAN. The primary requirement of NASTRAN is a large amount of core. NASTRAN can operate on as little as 45K words of core (UNIVAC 1108), but problem solving capability can be limited even with 65K words.[2] For large problems, at least 100K words should be available. The second question is whether sufficient auxiliary storage space is available. (This allocation can be provided through disk, drum, or other available storage devices.) A minimum of 30 files must be defined, each requiring a space allocation of 50K to 500K words.[3] As problem size increases, so does the need for auxiliary storage. For convenient operation, the hardware should be configured with at least five tape drives available to NASTRAN. If only a minimum hardware configuration is available, NASTRAN will operate, but it will be limited as to the size of a problem that it can solve. Moreover, a problem run on a larger system will usually take considerably longer to run on a smaller configuration, since insufficient core storage may cause spill in matrix operations. Other hardware features such as the number and speed of I/O channels and the speed of the auxiliary storage devices will also affect the operating efficiency of NASTRAN.

SOFTWARE IMPLEMENTATION

Source Language Creation

When we began to code NASTRAN and create source decks, it soon became apparent that card decks were too bulky. IBM 7094 IBSYS provided a card to tape utility, and NASTRAN card image source tapes were used. Compilations were made from the tape and the tape was updated via a CSC-written update program as new or modified decks were added.

[2]For example, a 45K system should be able to solve a 175-order complex eigenvalue problem; a 65K system, a 900-order problem; and a 100K system, a 2,100-order problem.

[3]For example, a single 1,000 order double precision matrix that is 25 percent dense requires 500,000 words of storage.

When the first NASTRAN system was to be developed on the UNIVAC 1108 EXEC II, a corresponding source tape was needed. A utility program to convert the IBM 7094 tape to UNIVAC 1108 EXEC II format was written. Once converted, this tape was acceptable as input to the FORTRAN compiler. Eventually, corresponding programs were written to provide input source tapes to the UNIVAC 1108 EXEC 8, the IBM 360, and the CDC 6600.

The first step in putting NASTRAN on a computer is to obtain or create a source tape that can be used as input to the sytem's compiler. In all likelihood, this will require creating a program that converts a current UNIVAC 1108, IBM 360, or CDC 6600 source tape to the required format. However, some operating systems may already have the capability to handle this task (the RCA SPECTRA 7, for example, accepts IBM 360 tape) and eliminate this step.

FORTRAN Compilation

The next step, and supposedly the easiest, is the compilation of all of the source language decks. Surprisingly, however, this step has activated numerous problems in the past. This situation no doubt has been due to the large number of source decks (700 subroutines, 145,000 source statements) and the variety and complexity of the code involved; if any errors exist in the target computer's FORTRAN compiler, they usually show up while the NASTRAN subroutines are being compiled. These errors appear either as a compilation abort, extraneous error messages, or incorrectly generated object code. Incorrect code may occur when optimization of the object code is selected. If this occurs, the use of optimization should be avoided. If other compiler errors are identified, they can usually be circumvented by rewriting the FORTRAN code.[4]

Another and potentially more serious problem relates to the definition of standard FORTRAN. CSC in attempting to avoid this problem defined a subset of the ASA standard FORTRAN IV that was to be used. This meant that certain features available under one compiler could not be used unless those features were available on all compilers. As a result, octal or hexadecimal data statement, for example, could not be used. However, in spite of our efforts, it was impossible to define a subset that was compatible on the CDC 6600. That system's compiler required a different format for nonstandard returns and did not allow multiple entry points.

[4]For example, it was found that EXEC 8 failed to compile properly when a deck with one entry point contained nonstandard returns. An additional dummy entry point was inserted into decks having one entry point and nonstandard returns, and correct code was thereafter generated.

When it became necessary to implement NASTRAN on the CDC 6600, a program was written to convert the source decks to 6600 compiler format. This program, similar to the FORTRAN II to FORTRAN IV SIFT program, modifies the format of nonstandard returns and eliminates multiple entry points.

With a strong background in working with FORTRAN, and with data on the integrity of the FORTRAN compiler involved, the degree of difficulty in compiling NASTRAN can be measured. If a given compiler is known to have errors, it should be avoided - presuming, of course, that a choice of compilers is available. If optimization has caused problems in the past, it should not be used. If the FORTRAN IV language differs from the standard that NASTRAN has used, a SIFT-type program may have to be developed. Overall, however, with a competent system, this task should be a purely mechanical one, consisting of a string compilation of all decks.

Source and Object Library Updating

Inherent in the compilation of source language decks is the creation of an object library tape. In order to maintain library tapes, a source and object library edit and update program is necessary. In the case of the IBM 7094, a source and object library edit and update program did not exist, and such a program therefore had to be written. Although a third generation system generally has such a capability, the capability is sometimes cumbersome to use. (For example, the IBM 360 object tape has to be edited to provide control section names for input to the Linkage Editor.)

By using library edit features, changes to source programs can be made by simply specifying in an update deck the changes that are to be made to the library source program. When this deck is compiled, the source and object libraries are automatically updated.

A sophisticated library edit capability in a target system can greatly facilitate compilation and subsequent updates to the system. Desirable features of an edit program include the following: (1) an update option for inserting, deleting, or adding cards, (2) the capability of allowing source and object libraries to be maintained on random access devices, (3) the ability to automatically update the source and object libraries when a deck is recompiled, and (4) the capability of listing and identifying all decks by time and date compiled.

Machine-Dependent Deck Conversion

Once the FORTRAN decks have been compiled, the machine-dependen programs must be converted. These machine-dependent programs consis of several assembly language decks, some Data Block decks, and some FORTRAN decks.

Through a concerted effort, assembly language coding was kept to a minimum, since CSC considered it more important to provide easily convertible FORTRAN code than to obtain the increase in efficiency provided by assembly code. The few assembly language programs that do exist consist of function-type subroutines (AND, OR, LSHIFT, etc.) and operating system-type interface routines (CPU clock, time of day clock, console message, etc.). Of the assembly language subroutines, some can be simply converted (i.e., AND, OR, LSHIFT), while others must be implemented according to the interface with the system (i.e., reading the CPU clock). Generally, the operating system provides subroutines that return the time of day, that return the elapsed CPU time, and that send messages to the computer operator. Under these circumstances, the only change required in the NASTRAN decks is to code a routine that calls the system subroutine providing these capabilities.

During NASTRAN's development, block data programs that contained system parameters were defined. These parameters include machine characteristics, and various subroutines utilize this data. In installing NASTRAN, the system parameters must be modified to identify such characteristics as computer word size, number of bits per character, number of characters per word, timing information, etc. The application subroutines utilize this information to accomplish such tasks as extracting a character from a word, based on the character and word length definitions contained in the data base.

The third set of decks that must be changed are the machine-dependent FORTRAN decks. These decks generally handle some of the NASTRAN-operating system interfaces. One of these decks is GINOIO, which contains the physical I/O calls to the system output devices. Preferably, GINOIO should contain macro-level I/O calls, but can contain FORTRAN READ/WRITE calls, although such calls are extremely inefficient. A second deck that must be modified (and that may have to be in assembly language) is GNFIAT, which must identify and transmit to NASTRAN the number and name of all files allocated to the run. Several smaller decks that accomplish initialization and link switching must also be changed.

To accomplish the conversion of these machine-dependent programs usually requires persons knowledgeable in both the intricacies of NASTRAN and the internal aspects of the operating system.

Absolute Program Creation

One of the most difficult tasks required in installing NASTRAN is the creation of the overlays and the generation of the absolute (executable) elements, since no two loaders of third generation computers are even remotely similar. In addition to this problem, most loaders are not equipped to handle an overlay as large or as complex as that of NASTRAN. Fairly early in the development of NASTRAN, we exceeded the capacity of the IBM 7094 loader, and subsequently exceeded a size limitation on the UNIVAC 1108 EXEC II due to fixed table sizes. However, even the third generation loaders have not been well equipped to handle the NASTRAN overlay.

NASTRAN is divided into 14 separate links, each with its own overlay tree structure. These links can be created independently and can be thought of as separate programs. Therefore, if a deck is changed in one link, only that link has to be regenerated. Communication between the links is handled via data contained on external storage. The EXEC 8 overlay loader handled the structuring of each link extremely well. Only those decks that defined origins had to be explicitly named, while all other subroutines were included automatically. However, EXEC 8 failed in providing a convenient mechanism for crossing between links. This was eventually solved by using external control cards to determine the link to be executed, calling the link in, and passing control to it. On the IBM 360, the linking concept is entirely different and consists of a super link occupying a region of its own. The functional links are loaded into a second region, with the super link program responsible for passing control between functional links.

In addition, the execution time loading characteristics between systems differ. The UNIVAC 1108 has a segment loader, which causes a segment of an overlay to be loaded only when the segment is explicitly called. The IBM 360, on the other hand, has a string loader, loading the segment called and all other segments between it and the main link. This caused problems; a job might run for example, on the IBM 360 but not the UNIVAC 1108 because the same segments were not loaded into core at the same time.

When we began the installation of NASTRAN on the CDC 6600, we encountered a loader with such limited capability that implementation was impossible. It was therefore necessary to write a loader to handle NASTRAN's requirements. A loader was written for the CDC 6600 (primarily in FORTRAN), and was patterned after the IBM 360 Linkage Editor/Loader.

The difficulty of establishing NASTRAN on a system depends on the capabilities of the system loader. If other large programs requiring overlays have been implemented successfully on the target computer, then NASTRAN can probably be installed. As mentioned earlier, if the operating system has a sophisticated loader capable of handling large overlays, the problems connected with NASTRAN installation will be minimal. If, however, the required loader capabilities do not exist, a new loader may have to be written. Since most third generation systems possess competent loaders, however, the likelihood of having to write a new loader is not great.

NASTRAN Execution

Once the absolute programs have all been created, NASTRAN is ready for execution. The control stream for executing NASTRAN must now be constructed by the user. NASTRAN requires a large number of separate I/O files (more than 30) with a significant amount of peripheral storage allocated to each file. These files are defined by means of the operating system's control language[5] and are identified by FORTRAN logical unit numbers.[6]

The operating system's control language is also used to specify the location of absolute programs (i.e., if they are catalogued on permanent storage) or to cause the programs to be read from tape. If sufficient auxiliary storage is available, the NASTRAN absolutes should remain resident, rather than be reloaded from tape for each run. The control stream must also identify under what abnormal end conditions a dump is to be taken, what portions of main core storage are to be dumped, and with what format.

This, of course will not be the first time in implementing NASTRAN on the target computer that extensive use has been made of the operating system's control language. Throughout the initial stages of NASTRAN generation, the compilations and library edits required the use of many control language features. From our experience, we found that a control language should be sophisticated enough to perform the necessary tasks, but not so detailed that it becomes cumbersome to use and difficult to understand.

Tailoring of NASTRAN to the Computer System

At this point, NASTRAN should be operational on the target computer. However, the installation should not yet be considered complete. Ideally, NASTRAN should be tailored to the particular computer system. This customizing includes setting up standard procedures to reduce the effort required to operate NASTRAN and extend its flexibility.

Many things can be done to simplify the control stream required to execute NASTRAN. One of these, already mentioned, is to catalog the NASTRAN system on a permanent file that is available to all users. Another means might be to group the many control cards necessary to define the files into one data set that is callable by a single control card. It may be possible to include in this call a

[5]For example, on the IBM 360 DD cards are used to define the unit number, physical device (tape, disk, etc.,) and space allocation.

[6]For example, the integers 8, 9, 10, 11, etc.

parameter that defines the space allocation.[7] It may also be possible to specify a region size on the run card, and therefore to provide a flexible NASTRAN system in terms of core and file size. By implementing the above suggestions, it is likely that the number of control cards needed can be reduced to as few as four or five.

Several changes can also be made to improve the run efficiency of NASTRAN. The NASTRAN buffer size can be tailored to fit the particular random access devices used. That is, buffer size can be set to correspond to half-track or full-track. Since data buffers should not cross track boundaries, such tailoring can improve the efficiency of the I/O and provide better utilization of auxiliary storage.

Machine timing characteristics such as internal arithmetic speed and I/O speed are stored in the data base, and are used to make logical decisions concerning the most efficient means of decomposing a matrix. Timing studies can be made to refine the data base times and thus to enable the decomposition routines to make proper decisions.

A longer range goal might include the conversion of some of the time-critical FORTRAN routines to assembly language, since conversion of these routines can lead to a substantial savings in machine time. Here, the I/O and PACK/UNPACK routines are good candidates for conversion. Also, if macro-level I/O was not used initially in GINOIO, it is well worth considering it at this point.

CONCLUDING REMARKS

The user who intends to install NASTRAN on a computer should now have a reasonable idea of the nature of the task before him. He also has an outline of the necessary steps to install NASTRAN. However, each computer is unique, and the problems as well as the solutions that he encounters in implementing NASTRAN may be distinctly different from those of the past.

Users about to install NASTRAN can reduce the magnitude of the installation effort if three resources are available: first, a capable and well checked out operating system; second, a person knowledgeable in the use and internal workings of that system; and third, a person experienced in the use and internal design of NASTRAN.

[7]On the IBM 360, for example, a PROC was set up to define the DD cards. A parameter included in the PROC defines the space allocation for each file.

REFERENCE

1. Second Quarterly Report for NASA General Purpose Structural Analysis Program (1 November 1966 - 1 January 1967) pp. 25-71, prepared by CSC for Goddard Space Flight Center, Greenbelt, Maryland.